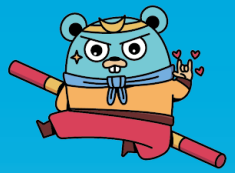


用Go构建高性能数据库中间件

徐成选

小米商城

xuchengxuan@xiaomi.com

自我介绍

- 15年初开始使用Go，被Go的生产力、性能所吸引
- 微服务
- 数据库、缓存中间件
- 其他一些偏业务基础服务，例如: 库存代理、人群服务、ID生成器等

Agenda

- Go in XiaoMi
- Gaea简介
- Why Go
- Gaea有关的几个技术点
- Impressive Runtime
- Go toolchains in Gaea
- Tests

Go in XiaoMi

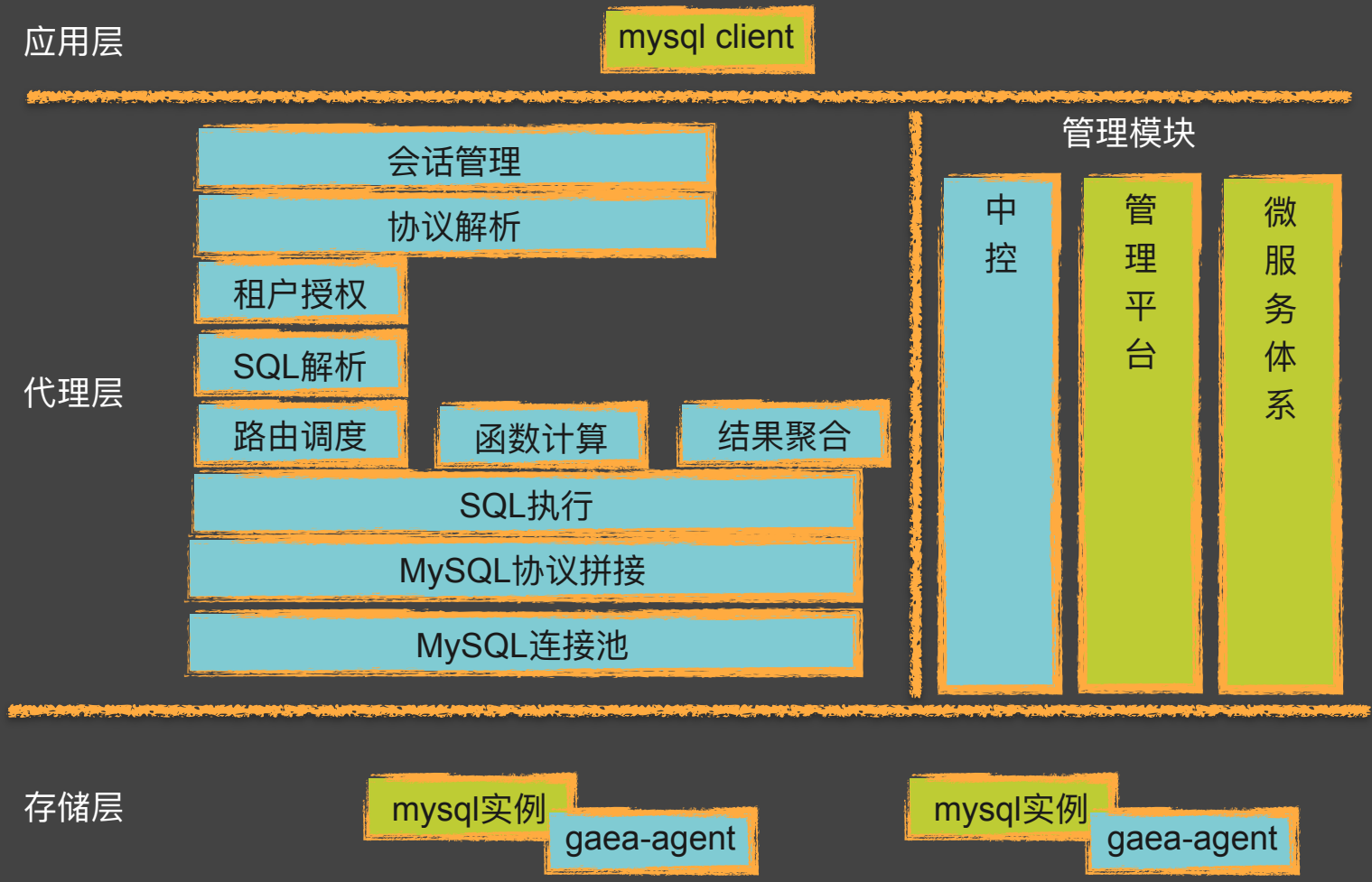
- 14年引入Go，最初解决日志收集问题，后来开发了大秒
- 商城、有品、金融、IoT、电视、云平台等部门
- 中间件、微服务体系、云计算、运维平台、业务系统等等
- 基于koala的微服务数百个，商城的后端系统都有Go的身影，包括订单、活动等非常核心的系统。

Gaea背景

- 内部mycat黑盒使用、不能及时定位问题
- 连接超时、Load过高、内存溢出
- 手工书写多种配置，易出错、难管理
- 历史包袱: 多个代理，缺乏平台型方案

Gaea特性

- 分库分表，兼容mycat、kingshard路由方案
- Prepared Statements(分库分表)
- 读写分离，多个从实例负载均衡
- 多租户，租户之间软隔离
- 错误、慢sql指纹
- 配置热加载
- 连接池
- 使用TiDB sqlparser


应用层
mysql client
代理层
会话管理
协议解析
租户授权
SQL解析
路由调度
函数计算
结果聚合
SQL执行
MySQL协议拼接
MySQL连接池
管理模块
中控
管理平台
微服务体系
存储层
mysql实例
gaea-agent
mysql实例
gaea-agent

成果

- 18年11月份上线、2套集群、16个非分片业务、2个分片业务
- QPS提升25%
- 并发优秀，不再被短连接打爆
- 服务平台化，业务申请=>DBA分配=>租户生效
- prometheus+grafana监控

Why Go

- 并发友好: one connection per goroutine
- 开发效率高
- 工具丰富
- kingshard、vitess、tidb等优秀项目
- 团队Go经验比较丰富，也一直很喜欢用Go开发系统

配置热加载

方案一

```
configAtomic atomic.Value
```

定义

```
config, err := NewConfig(confType)
config.Load()
p.configAtomic.Store(&configWrapper{config})
```

加载/reload

```
c := p.configAtomic.Load()
if c == nil {
    err = fmt.Errorf("invalid remote config instance")
    return
}

configWrap, ok := c.(*configWrapper)
if !ok {
    err = fmt.Errorf("invalid remote config instance, type not xconf
    return
}

return configWrap.config, nil
```

获取配置

方案二

定义

```
switchIndex util.BoolIndex
namespaces  [2]*NamespaceManager
```

prepare

```
current, other, _ := m.switchIndex.Get()

// reload namespace prepare
currentNamespaceManager := m.namespaces[current]
newNamespaceManager := ShallowCopyNamespaceManager(currentNamespaceManager)
if err := newNamespaceManager.RebuildNamespace(namespaceConfig); err != nil {
    log.Warn("prepare config of namespace: %s failed, err: %v", name, err)
    return err
}
m.namespaces[other] = newNamespaceManager
```

commit

```
current, _, index := m.switchIndex.Get()

currentNamespace := m.namespaces[current].GetNamespace(name)
if currentNamespace != nil {
    go currentNamespace.Close(true)
}

m.switchIndex.Set(!index)
```


资源抽象

```go
// NewNamespace init namespace
func NewNamespace(namespaceConfig *models.Namespace) (*Namespace, error) {
    var err error
    namespace := &Namespace{
```

New

```go
// Close recycle resources of namespace
func (n *Namespace) Close(delay bool) {
    var err error
```

Close

```go
currentNamespace := m.namespaces[current].GetNamespace(name)
if currentNamespace != nil {
    go currentNamespace.Close(true)
}
```

Delay Close

连接池

- [cap, max]范围内自动调整容量
- 保活
- 超时获取

自动调整

定义

```
// ConnectionPool means connection pool with specific addr
type ConnectionPool struct {
    mu          sync.RWMutex
    connections *util.ResourcePool

    addr     string
    user     string
    password string
    db       string

    charset     string
    collationID mysql.CollationID

    capacity    int // capacity of pool
    maxCapacity int // max capacity of pool
    idleTimeout time.Duration
}
```

```
cp.connections = util.NewResourcePool(cp.connect, cp.capacity, cp.maxCapacity, cp.idleTimeout)
return
```

```
rp := &ResourcePool{
    resources:   make(chan resourceWrapper, maxCap),
    factory:     factory,
    available:   sync2.NewAtomicInt64(int64(capacity)),
    capacity:    sync2.NewAtomicInt64(int64(capacity)),
    idleTimeout: sync2.NewAtomicDuration(idleTimeout),
}
for i := 0; i < capacity; i++ {
    rp.resources <- resourceWrapper{}
}

if idleTimeout != 0 {
    rp.idleTimer = timer.NewTimer(idleTimeout / 10)
    rp.idleTimer.Start(rp.closeIdleResources)
}
```

主动关闭

```
type resourceWrapper struct {
    resource Resource
    timeUsed time.Time
}
```

时间戳

保活

- Ping ?
  - 多租户，需要Ping的连接很多
  - 配置经常变化，CPU消耗不可控
- 主动关闭+Retry
  - idleTimeout < mysql wait-timeout
  - reconnect not get again


```
// retry 3 times, close dc's conn、reset dc's stats and reconnect
for i := 0; i < 3; i++ {
    dc.Close()
    e := dc.connect()
    if e == nil { // no need to write data again and ephemeral buffer is recycled
        break
    }
}
```


Context

- 通过context实现超时获取连接
- context timeout范式
  - 函数定义context参数就可以超时取消吗?
  - 一定有一个goroutine去单独发送请求、接受应答
  - 一定有一个select去判断resp、context的状态

```go
func test2(ctx context.Context) {
    fmt.Println(ctx.Value("k1"))
    resp := make(chan struct{}, 1)
    // 处理逻辑
    go func() {
        // 处理耗时
        time.Sleep(time.Second * 10)
        resp <- struct{}{}
    }()

    // 超时机制
    select {
    case <-ctx.Done(): // 超时
        fmt.Println(ctx.Err())
        fmt.Println("timeout")
    case v := <-resp: // 正常应答
        fmt.Println("test2 function handle done")
        fmt.Printf("result: %v\n", v)
    }
    fmt.Println("test2 finish")
    return

}
```


Context是什么

- context是树? 是map?
  - 一系列实现 Context接口的非导出struct
  - 存储parent context或children context
  - 通过 AfterFunc或goroutine执行超时动作
  - valueCtx只有parent context，所以自顶向下传值

会话管理

- SetReadDeadline、SetWriteDeadline
  - 分散设置
  - 每一次读、写都需要设置，高频操作
  - 耗CPU
  - 线上海量连接带来性能问题

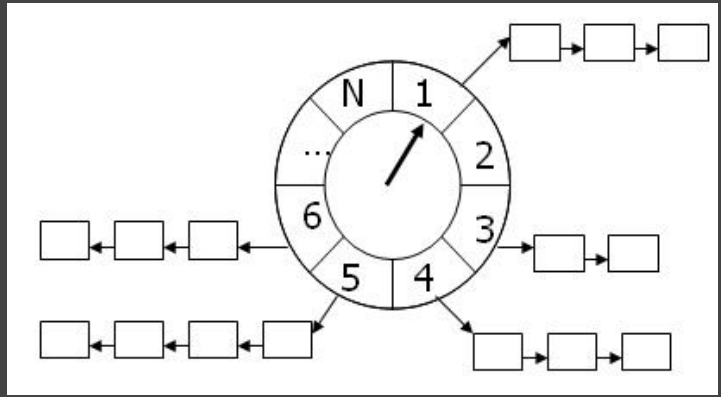
会话管理-方案1

- 定时设置
  - 可以缓解性能问题
  - 有一定误差、写法繁琐
  - 分散管理

```
readTick := time.NewTicker(c.ReaderTimeout / 2)
writeTick := time.NewTicker(c.WriterTimeout / 2)
defer readTick.Stop()
defer writeTick.Stop()
for !c.stop {
    select {
    case <-readTick.C:
        if c.readFlag {
            if err := c.Sock.SetReadDeadline(time.Now().Add(c.ReaderTimeout)); err != nil {
                errors.Trace(err)
            }
            c.readFlag = false
        }
    case <-writeTick.C:
        if c.writeFlag {
            if err := c.Sock.SetWriteDeadline(time.Now().Add(c.WriterTimeout)); err != nil {
                errors.Trace(err)
            }
            c.writeFlag = false
        }
    }
}
c.Sock.Close()
```


会话管理-方案2

- 时间片轮转
  - 集中式管理
  - CPU消耗极低



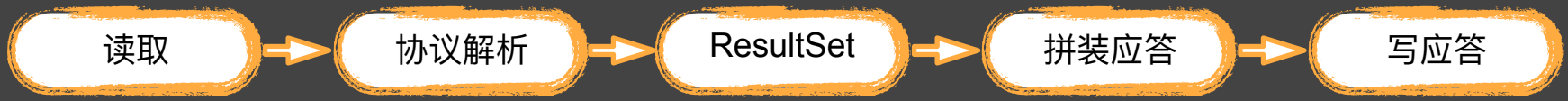

```
// added into time wheel
s.tw.Add(s.sessionTimeout, cc, cc.Close)
```

```
cc.Close()
cc.proxy.tw.Remove(cc)
```


GC? 内存

- GC暂停不存在问题，内存一直增长很慌

GC? 内存



header: make([]byte, 4)

body: make([]byte, len)

return body

data: make([]byte, len)

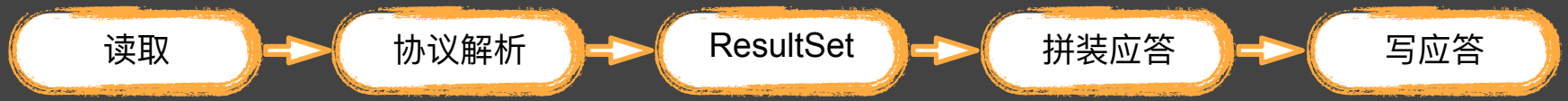
data=append(data, itemBytes)

.

.

.

sync.Pool




```
data, err := cc.c.ReadEphemeralPacket()
Execute(data)
cc.c.RecycleReadPacket()
```

```
data := cc.StartEphemeralPacket(length)
pos = WriteLenEncInt(data, pos, value)
pos = WriteUint16(data, pos, value)
                .
                .
                .
cc.WriteEphemeralPacket()
```

```
cc.StartWriterBuffering()
write…
err = cc.Flush()
```


[]byte pool改变流程

- 处理流程
  - Read->Execute(Response Everywhere)->Final Response

Read->Execute(return r)->Final Response

- 生命周期
  - 申请、回收操作规划到一个函数内

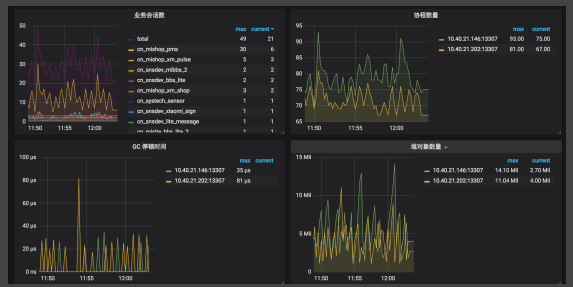
优化后效果

Go Runtime

- wrap pprof into http admin server
- runtime.MemStats、 NumGoroutine to prometheus
- go-torch for Flam Graph

Go Runtime

程序md5	内存占用	日志级别	权限认证	启动时间	来源	添加时间	服务主机管理操作
6551e0444bcc38f8fc80	512.13M	Debug	已关闭	2019-03-08 18:11	服务树同步	2018-11-19 19:18	cpu mem 火焰图
6551e0444bcc38f8fc80	2052.46M	Notice	已关闭	2019-03-08 18:07	服务树同步	2019-01-24 16:40	cpu mem 火焰图
6551e0444bcc38f8fc80	1538.10M	Notice	已关闭	2019-03-08 18:06	服务树同步	2019-01-24 16:40	cpu mem 火焰图



Go Toolchains

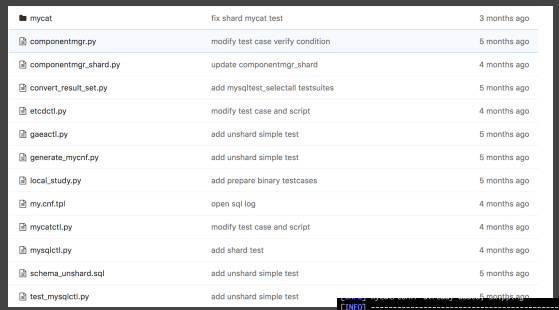
- glide
- git hook -> pre-commit
  - gofmt
  - golint
  - goimports
  - govet
  - godoc

Unit Tests

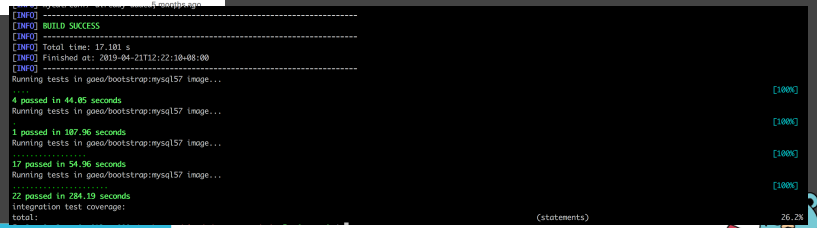
- 拆分有状态和无状态模块
  - 通过interface抽象服务，测试对应组件
- 测试覆盖报告
  - go test -coverprofile=.coverage.out ./…
- CI
  - commit/merge request -> gitlab pipeline -> all unit tests

Integration Tests

- 微型集成测试环境
  - 抽取mysql测试语句，增加分库分表场景，形成python脚本
  - 通过docker打包gaea、gaea-cc、etcd、mysql打造完整运行环境



mycat	fix shard mycat test	3 months ago
componentmgr.py	modify test case verify condition	5 months ago
componentmgr_shard.py	update componentmgr_shard	4 months ago
convert_result_set.py	add mysqltest_selectall testsuites	5 months ago
etcdctl.py	modify test case and script	4 months ago
gaeactl.py	add unshard simple test	5 months ago
generate_mycnf.py	add unshard simple test	5 months ago
local_study.py	add prepare binary testcases	5 months ago
my.cnf.tpl	open sql log	4 months ago
mycatctl.py	modify test case and script	4 months ago
mysqlctl.py	add shard test	4 months ago
schema_unshard.sql	add unshard simple test	5 months ago
test_mysqlctl.py	add unshard simple test	5 months ago



```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 17.101 s
[INFO] Finished at: 2019-04-21T12:22:10+08:00
[INFO] ------------------------------------------------------------------------
Running tests in gaea/bootstrap:mysql57 image...
                                                                        [100%]
4 passed in 44.05 seconds
Running tests in gaea/bootstrap:mysql57 image...
                                                                        [100%]
1 passed in 107.96 seconds
Running tests in gaea/bootstrap:mysql57 image...
                                                                        [100%]
17 passed in 54.96 seconds
Running tests in gaea/bootstrap:mysql57 image...
                                                                        [100%]
22 passed in 284.19 seconds
integration test coverage:
total:                                  (statements)            26.2%
```

Thanks