IT大咖说
知识分享平台

用Go语言实现推送服务器

陈叶皓

议程

- 推送服务器介绍
- Golang特点
- 推送服务架构
- 部分代码
- 上线效果

IT大咖说
知识分享平台

议程

- 推送服务器介绍
- 推送服务架构
- 部分代码
- 上线效果

什么是推送服务器

- 推送业务信息到手机端
- 始终保持连接

推送服务器要求

- 高并发
- 可靠性
- 高性能
- 支持水平扩展
- 无单点故障

Go语言特性

- 静态的、编译的
- 自动内存回收
- 命令式编程
- 函数可以作为值
- 面向并发
- 内置RPC支持

推送服务器要求的应对

- 高并发
  - goroutine
- 可靠性
  - 使用Redis暂存消息
- 高性能
  - 静态编译语言
- 支持水平扩展
  - 使用RPC组成集群
- 无单点故障
  - 使用Redis实现数据共享

Go语言的并发模型

- 事件驱动，共享线程池

  runtime.GOMAXPROCS(runtime.NumCPU())

- 使用“go”命令创建goroutine

  go sockstore.Start()

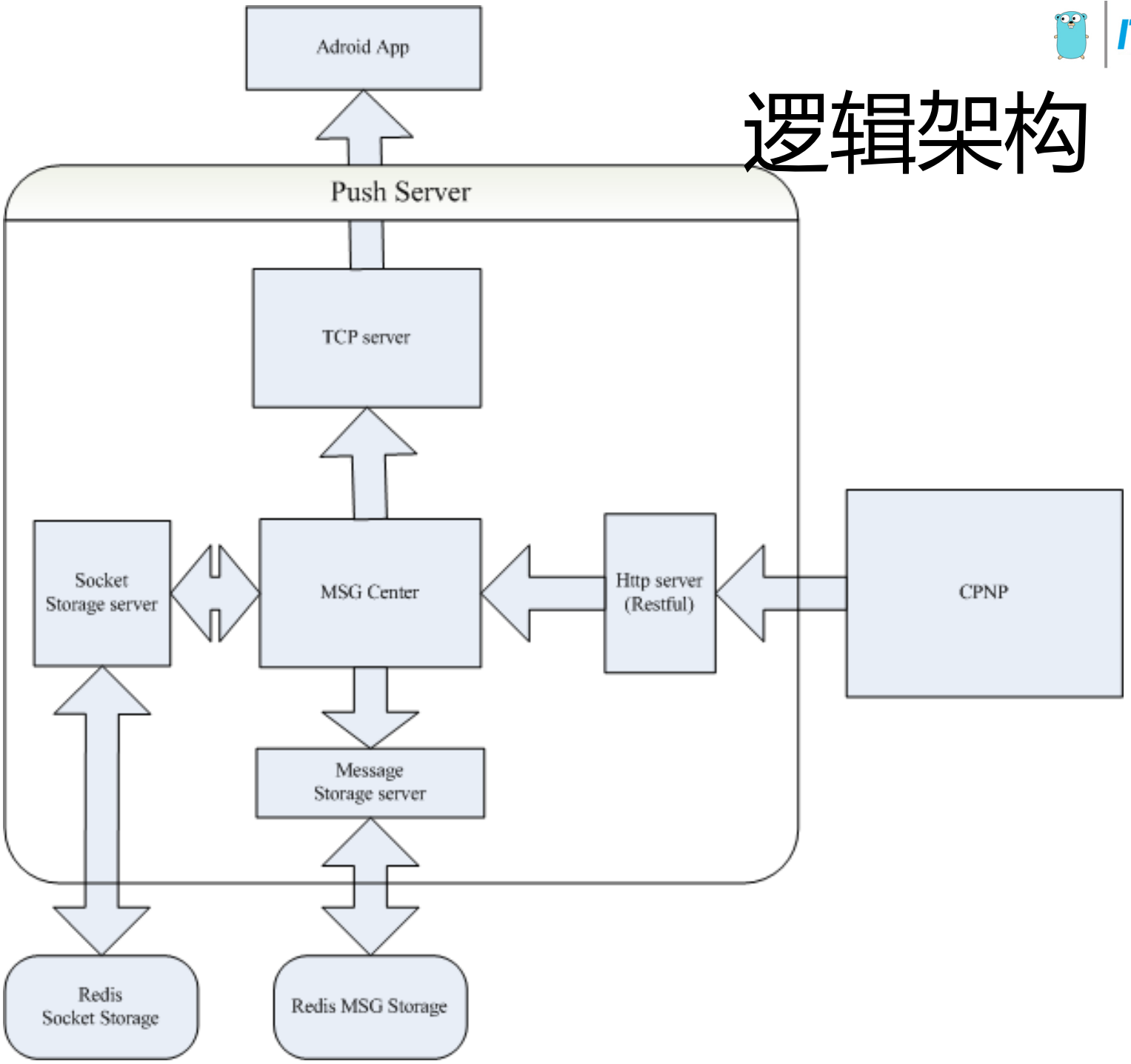
- goroutine使用channel交换消息
  - 异步场景，直接往指定channel发送数据
  - 同步场景，往channel发送的数据中，包含一个获取返回值的channel

IT大咖说
知识分享平台

议程

- 推送服务器介绍
- 推送服务架构
- 部分代码
- 上线效果

逻辑架构


Adroid App
Push Server
TCP server
Socket
Storage server
MSG Center
Http server
(Restful)
CPNP
Message
Storage server
Redis
Socket Storage
Redis MSG Storage

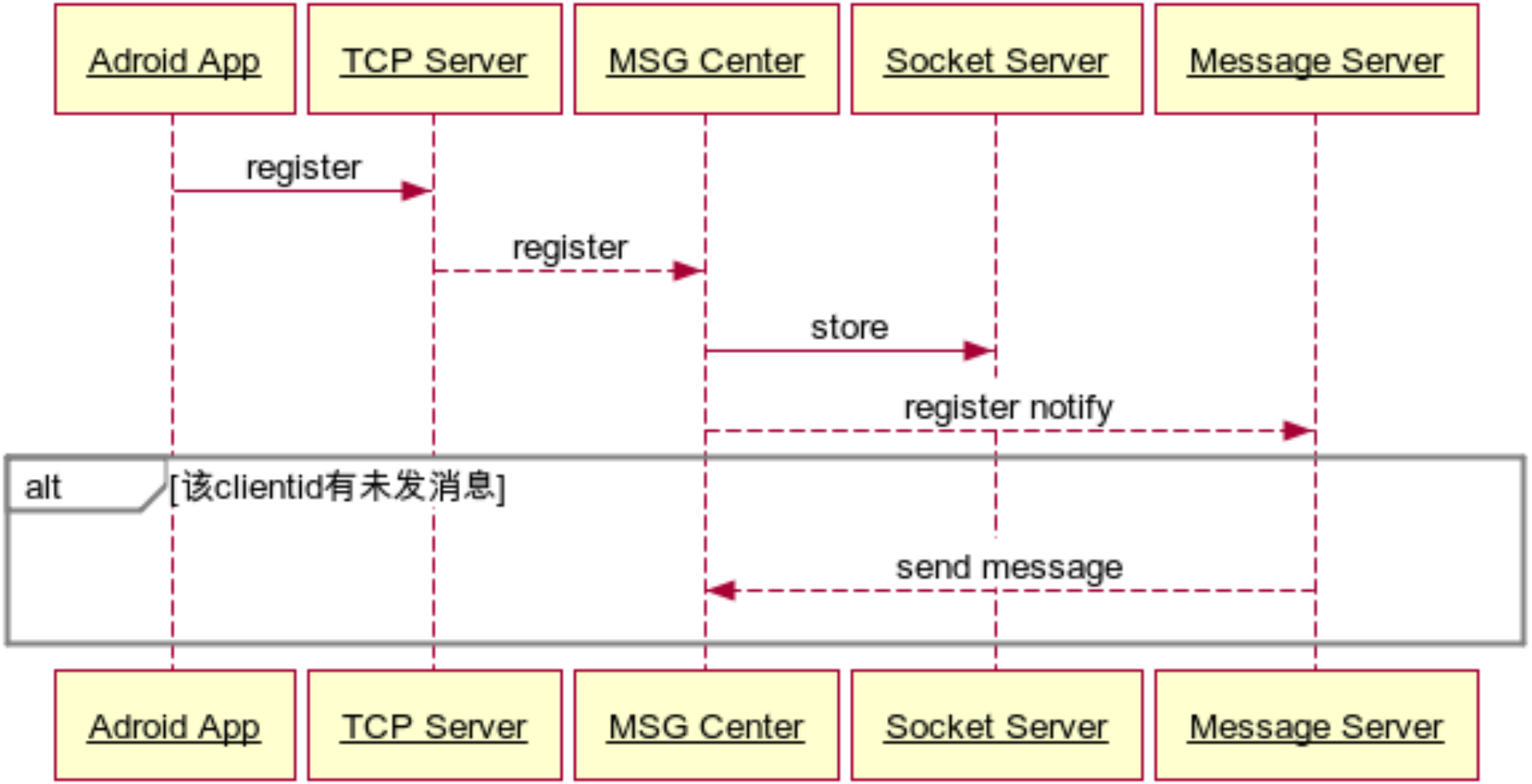
去中心化设计

- 客户端随机连接
- Redis集中存储地址表
- 信息发送2跳到达

消息缓存设计

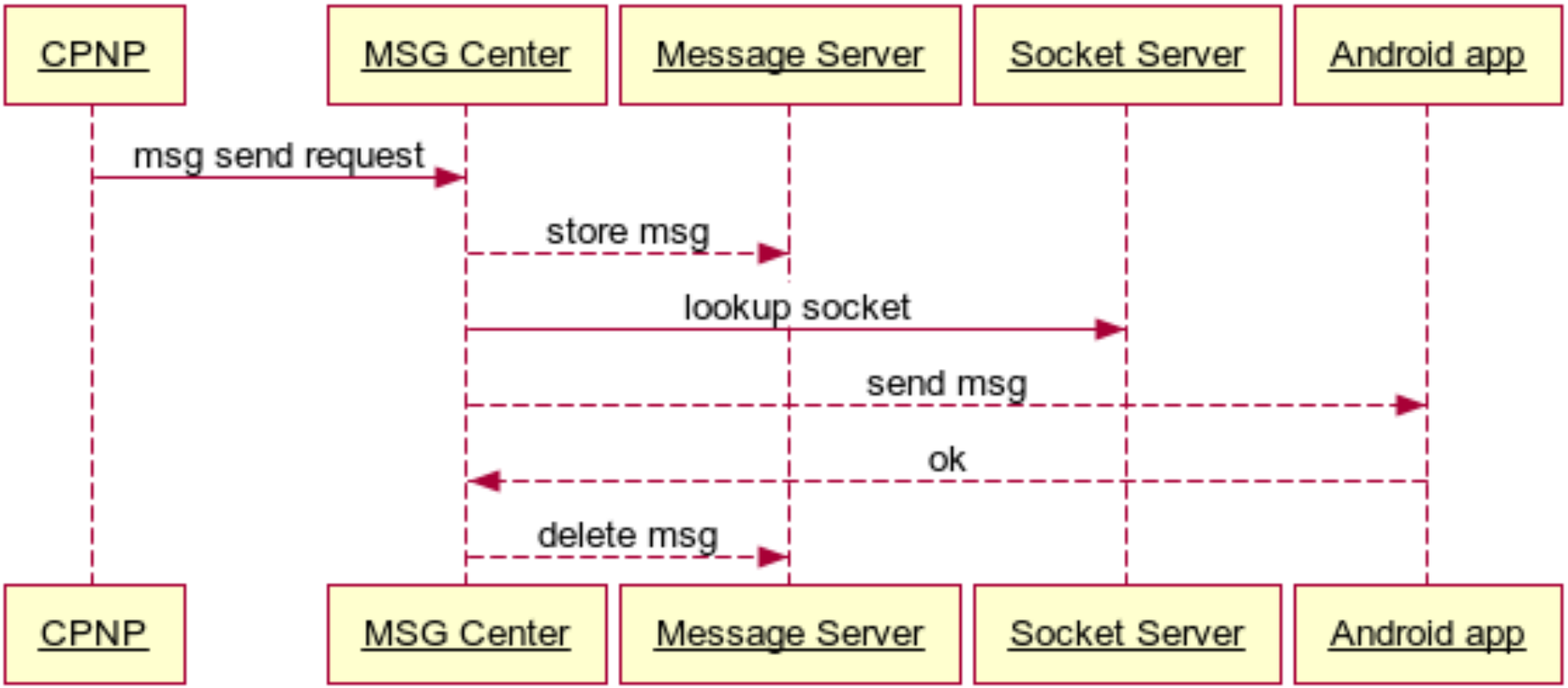
- 消息预存（Redis）
- 尝试发送
- 发送成功后删除

Adroid client register



www.websequencediagrams.com

Server Push



www.websequencediagrams.com

IT大咖说
知识分享平台

议程

- 推送服务器介绍
- 推送服务架构
- 部分代码
- 上线效果

串行场景-Socket Server

- 在主程序启动时启动
- 所有请求都用一个goroutine响应
- 对外提供API，实质是往goroutine发送消息

```
for {
    select {
    case value := <-socketMapChans.chGet:
        l4g.Debug("channelsvr.Start():chGet:", value.Key)
        value.ReplyChannel <- data[value.Key]
    case value := <-socketMapChans.chRegister:
        l4g.Debug("channelsvr.Start():chRegister:", value.clientId)
        data[value.clientId] = value.controller
    case value := <-socketMapChans.chDelete:
        l4g.Debug("channelsvr.Start():chDelete:", value)
        delete(data, value)
    case value := <-socketMapChans.chBroadcast:
        l4g.Debug("channelsvr.Start():chBroadcast:", value)
        for _, v := range data {
            v.Send("100000000", value)
        }
    case reply := <-socketMapChans.chGetCount:
        l4g.Debug("channelsvr.Start():chGetCount")
        reply <- len(data)
    }
}
```

并行场景-TCP Server

- 在有socket连接时创建
- 为每个socket创建一个goroutine
- 用心跳维持，超时关闭socket，同时退出goroutine
- 用全局字典，查找clientID对应的socket


```go
func StartTcpSvr() {
    onceTcpSvr.Do(func() {
        port := ":" + configsvr.Fetch("tcpsvr_port")
        if ":" == strings.TrimSpace(port) {
            port = ":9988" //default 9988
        }
        l4g.Debug("tcpsvr listen on port:%s", port)
        l, err := net.Listen("tcp", port)
        if err != nil {
            l4g.Error("Failure to listen: %s", err.Error())
            return
        }
        l4g.Debug("tcpsvr::listening......")

        for {
            l4g.Debug("Started!")
            if c, err := l.Accept(); err == nil {
                l4g.Debug("Accept()::", c)
                go StartChild(c) //new thread
            }
        }
        l4g.Debug("Everything is done!")

    })
}
```

```
func StartChild(c net.Conn) {
    l4g.Debug("tcpsvr::StartChild()::Started!")
    var clientRData *proto.TDatagram = nil
    defer func() {
        if nil != clientRData {
            Logout(clientRData.GetClientId())
        } else {
            l4g.Debug("clientRData is nil!")
        }
        c.Close()
        l4g.Debug("tcpsvr::StartChild()::Ended!")
    }()

    controller := &tTPCChans{
        chSend: make(chan *TChanSend),
        chQuit: make(chan bool),
    }

    go func() { // Read
        //l4g.Debug("tcpsvr::StartChild()::Read()::begin!()")
        reader := bufio.NewReaderSize(c, 256)
        ...
        controller.Quit()
    }()
```

```
    // Write
    //l4g.Debug("tcpsvr::StartChild()::Write()::begin!()")
    writer := bufio.NewWriterSize(c, 256)
WriteLoop:
    for {
        select {
        case value := <-controller.chSend:
            msgD := proto.MakeMsgDatagram(string(clientRData.ClientID[:20]),
                value.MsgId, value.MsgBody)
            if "" != msgD {
                //l4g.Debug("Write:msg:", msgD)
                _, err := writer.Write([]byte(msgD))
                if err != nil {
                    l4g.Error("Failure to write:", err)
                    break WriteLoop
                }
                writer.Flush()
                //l4g.Debug("Write:", value)
            }
        case <-controller.chQuit:
            //l4g.Debug("StartChild()::received a quit message!")
            break WriteLoop
        }
    }
    //l4g.Debug("tcpsvr::StartChild()::Write()::end!()")
}
```

议程

- 推送服务器介绍
- 推送服务架构
- 部分代码
- 上线效果

比较

	.net push server	Go push server
操作系统	Windows server	CentOS Linux
服务器数量（同配置）	16	8
单台服务器支持连接数	20万	80万
Cpu平均占用	10%到30%，波动频繁	稳定在5%
运维需求	使用脚本每天重启一次	稳定运行数月未崩溃

IT大咖说
知识分享平台

Q&A

Thanks