C/C++Linux服务器开发

高级架构师课程

三年课程沉淀 | 五次精益升级 | 十年行业积累 | 百个实战项目 | 十万内容受众

讲师:darren/326873713

扫一扫 升职加薪

班主任:柚子/2690491738

讲师介绍--专业来自专注和实力

King老师

系统架构师，曾供职著名创业公司系统架构师，微软亚洲研究院、创维集团全球研发中心。国内第一代商业Paas平台开发者。著有多个软件专利，参与多个开源软件维护。在全球化，高可用的物联网云平台架构与智能硬件设计方面有丰富的研发与实战经验。

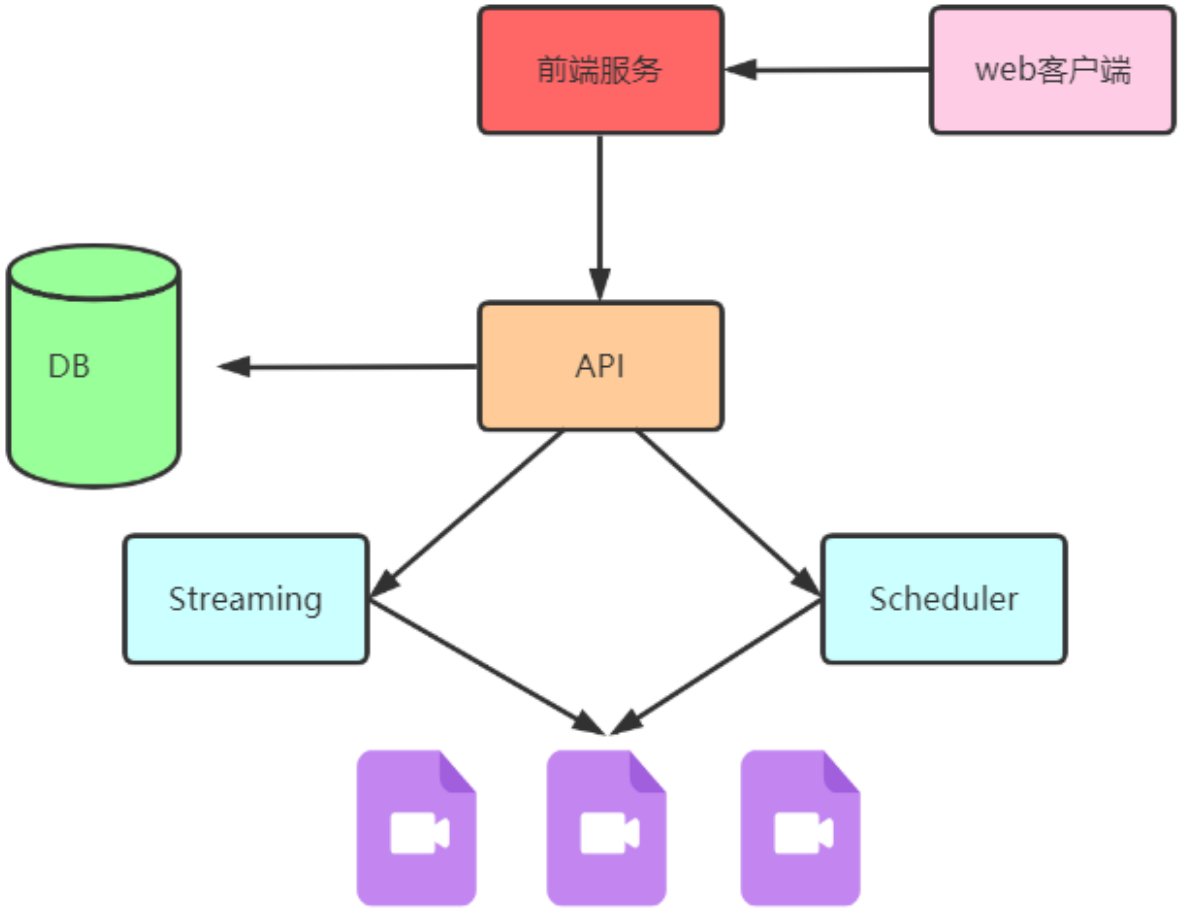
Darren老师

曾供职于国内知名半导体公司（珠海扬智/深圳联发科），曾在某互联网公司担任音视频通话项目经理。主要从事音视频驱动、多媒体中间件、流媒体服务器的开发，开发过即时通讯+音视频通话的大型项目，在音视频、C/C++/GO Linux服务器领域有丰富的实战经验。

3 stream-scheduler-web详细设计

1. streamserver设计
2. scheduler设计
3. web设计

0 总体架构



1 Streamserver

- 静态视频，
- 独立的服务，可独立部署
- 统一的API格式

1.1 Stream Server-对外接口

- /videos/:vid-id -> streamHandler 文件播放
- /upload/:vid-id -> uploadHandler 文件上传

1.2 代码整体设计

- 流控机制
- middleware的作用

1.3 流控机制-token bucket

- 为什么需要流控
- 拿到token才能继续进一步处理
- 为什么不用数组
  - go routine是并发的，如果用变量则需要加锁
  - go处理并发用channel

limiter.go

1.4 在http middleware加入流控

```
type middleWareHandler struct {
    r *httprouter.Router
    l *ConnLimiter
}
```

```
func NewMiddleWareHandler(r *httprouter.Router,
 cc int) http.Handler {
    m := middleWareHandler{}
    m.r = r
    m.l = NewConnLimiter(cc) // 限制数量
    return m
}
```

1.5 stream handler的实现

- streamHandler 读取文件播放
- uploadHandler 上传文件

2 Scheduler调度器

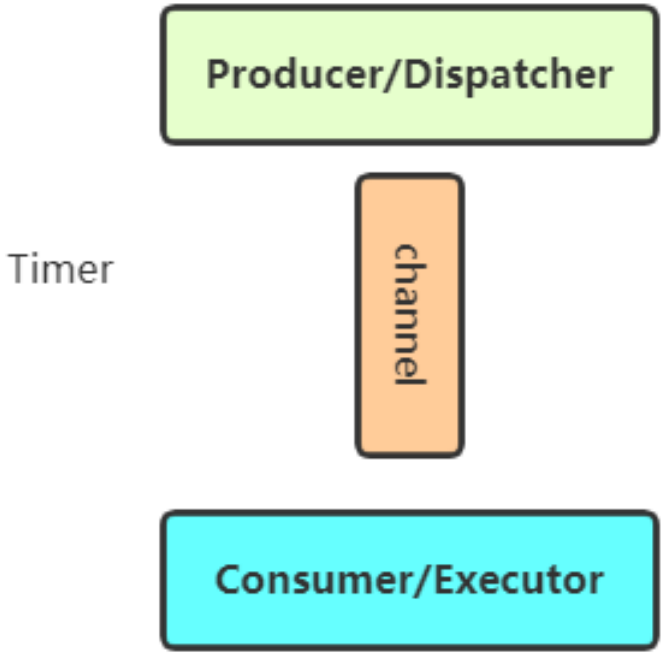
- 什么是scheduler
- 为什么需要scheduler
- scheduler通常做什么

异步任务、延时任务、定时任务

2.1 Scheduler包含什么

- REST ful 的HTTP server
- Timer
- 生产者消费者模型下的task runner

2.2 Scheduler架构



2.3 代码架构

- dbops 数据库查询和删除
- taskrunner 执行任务
  - runner.go 处理任务流程（生产消费模型）
  - tasks.go　执行任务（具体的生产、消费）
  - trmain.go 实现定时任务，比如每3秒执行一次
- handlers.go 处理api
- main.go程序入口
- response.go http响应封装

2.4 task实现

主要实现任务的处理流程和执行具体任务

- Controller 流程控制channel
- Error 错误控制channel
- Data 真正的任务数据channel

```
type Runner struct {
    Controller controlChan
    Error      controlChan
    Data       dataChan
    dataSize   int
    longLived  bool
    Dispatcher fn
    Executor   fn
}
```

runner.go

tasks.go

2.5 timer实现

通过定时器实现定时任务

```
type Worker struct {
    ticker *time.Ticker
    runner *Runner
}
```

trmain.go

```
func NewWorker(interval time.Duration, r *Runner) *Worker {
    return &Worker{
        ticker: time.NewTicker(interval * time.Second),
        runner: r,
    }
}

func (w *Worker) startWorker() {
    for {
        select {
        case <-w.ticker.C: // 时间到了
            go w.runner.StartAll()
        }
    }
}
```

3 web前端服务

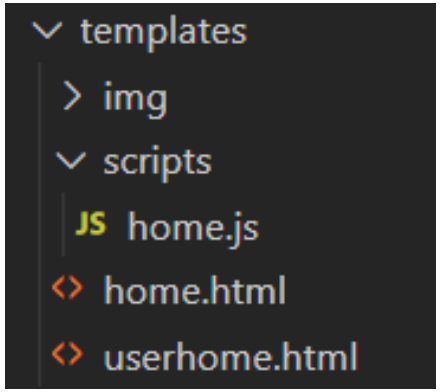
- Go模板引擎
- API处理
  - API透传
  - proxy代理

3.0 代码架构

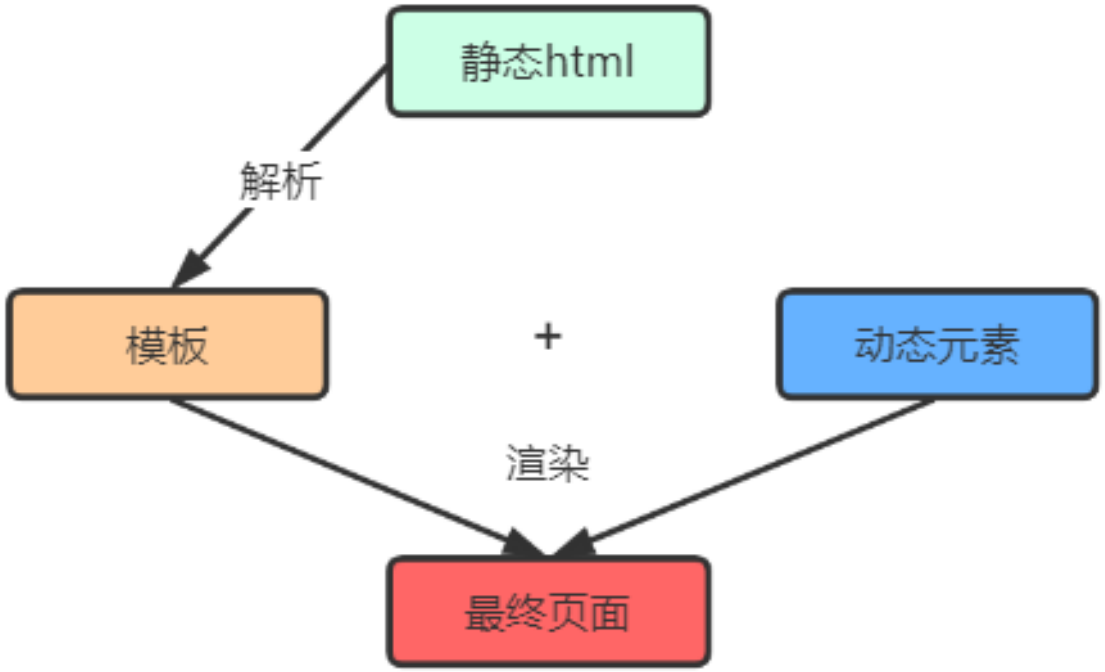
- templates html模板
- client.go 处理api透传
- defs.go 结构体定义
- handlers.go api入口处理函数
- main.go 主入口

3.1 Go的模板引擎

- 模板引擎是将HTML解析和元素预置替换生成最终页面的工具
- Go的模板有两种text/template和html/template
- Go的模板采用动态生成的模式



3.2 Go的模板引擎-渲染流程



3.3 页面渲染

- 主页渲染：homeHandler
- 用户页渲染：userHomeHandler

3.4 api透传模块实现

- apiHandler 处理逻辑分析

3.5 proxy转发的实现

- proxyHandler处理逻辑非分析