GoMall 高并发分布式电商秒杀系统

项目简介

GoMall 是一个从**单体架构逐步演进到微服务架构**的电商实战项目，专注于解决高并发场景下的秒杀难题。项目完整实现了电商核心业务模块，并重点攻克了**超卖、少卖、高并发**等技术痛点。

核心特性

特性	实现方案	效果
高性能	Gin + Redis 缓存 + Lua 原子脚本	支撑万级 QPS 秒杀
高可用	RabbitMQ 异步削峰 + 优雅关闭	系统稳定运行
可观测	OpenTelemetry + Prometheus + Zap	全链路监控
可扩展	模块化设计 + 服务注册发现	支持微服务拆分
统一规范	标准化响应 + 参数校验 + 错误码	前后端高效协作
支付能力	微信支付（沙箱）	支付流程完整跑通

技术栈

后端技术

技术	版本	用途
Go	1.23+	后端开发语言
Gin	v1.9.1	HTTP Web 框架
GORM	v1.25.5	MySQL ORM
gRPC	v1.60.1	微服务通信
Viper	v1.18.2	配置管理
JWT	-	认证授权
bcrypt	-	密码加密

前端技术

技术	版本	用途
React	18+	UI 框架
TypeScript	5+	类型安全
Vite	5+	构建工具
Axios	-	HTTP 客户端
Zustand	-	状态管理
React Router	-	路由管理

中间件

技术	版本	用途
MySQL	8.0+	持久化存储
Redis	7.0+	缓存、分布式锁、计数器

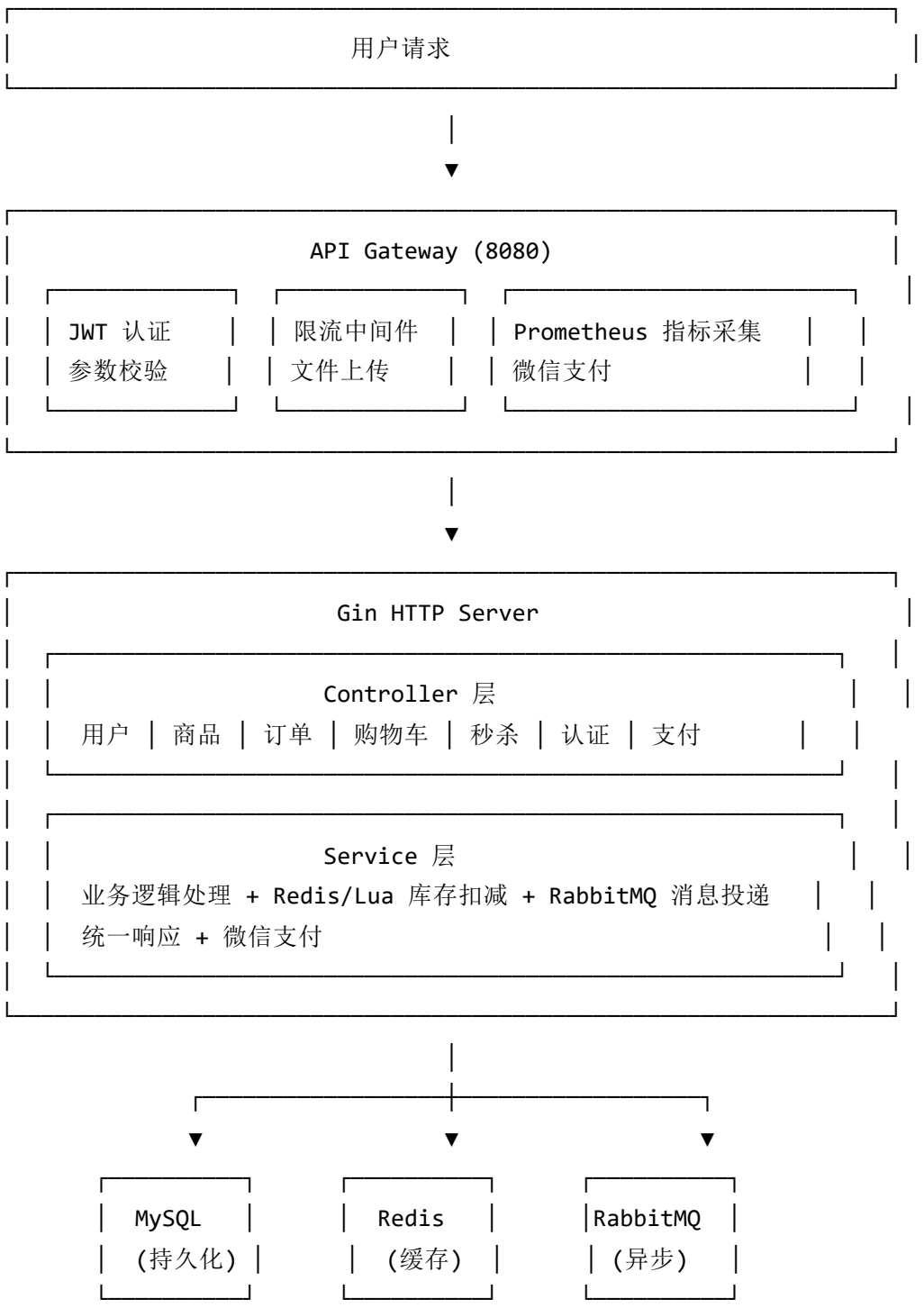
技术	版本	用途
RabbitMQ	3.12+	流量削峰、异步解耦

运维监控

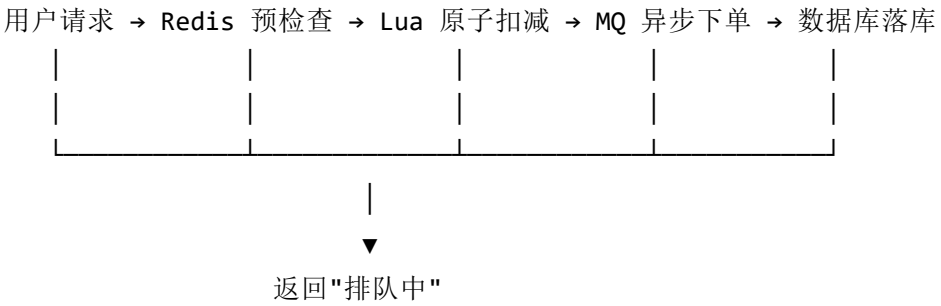
技术	用途
OpenTelemetry + Jaeger	分布式链路追踪
Prometheus	指标监控
Swagger	API 文档
Uber Zap	结构化日志
Docker	容器化部署

架构设计

系统架构图


用户请求
API Gateway (8080)
JWT 认证
参数校验
限流中间件
文件上传
Prometheus 指标采集
微信支付
Gin HTTP Server
Controller 层
用户 | 商品 | 订单 | 购物车 | 秒杀 | 认证 | 支付
Service 层
业务逻辑处理 + Redis/Lua 库存扣减 + RabbitMQ 消息投递
统一响应 + 微信支付
MySQL
(持久化)
Redis
(缓存)
RabbitMQ
(异步)

秒杀核心流程



支付流程（微信沙箱）

1. 前端调用统一下单 → 后端调用微信API
2. 返回支付二维码链接
3. 用户扫码支付 → 微信回调通知
4. 后端验证签名 → 更新订单状态
5. 前端轮询订单状态 → 显示支付成功

目录结构

```
gomall/
├── backend/                    # 后端服务 (Go + Gin)
│   ├── main.go                 # 程序入口
│   ├── Makefile                # 构建脚本
│   ├── Dockerfile              # Docker 构建文件
│   ├── go.mod / go.sum         # Go 依赖管理
│   ├── conf/                   # 配置文件
│   │   ├── config.yaml         # 默认配置
│   │   ├── config-dev.yaml     # 开发环境配置
│   │   └── config-prod.yaml    # 生产环境配置
│   ├── deploy/                 # 部署配置
│   │   ├── docker-compose.yml      # Docker Compose (单体架构)
│   │   ├── docker-compose-microservices.yml  # Docker Compose (微服务)
│   │   └── mysql/
│   │       └── init.sql        # 数据库初始化脚本
│   ├── internal/               # 内部业务代码
│   │   ├── api/                # HTTP Handlers (Controller层)
│   │   │   ├── handler.go              # 用户/商品/订单处理器
│   │   │   ├── cart_handler.go         # 购物车
│   │   │   ├── seckill_handler.go      # 秒杀
│   │   │   ├── auth_handler.go         # 认证（JWT刷新/改密/退出）
│   │   │   ├── file_handler.go         # 文件上传
│   │   │   ├── wechat_pay_handler.go   # 微信支付
│   │   │   └── health_check.go         # 健康检查
│   │   ├── config/             # 配置加载
│   │   ├── database/           # MySQL 连接
│   │   ├── gateway/            # API 网关
│   │   ├── grpc/               # gRPC 服务
│   │   ├── logger/             # Zap 日志
│   │   ├── metrics/            # Prometheus 指标
│   │   ├── middleware/         # 中间件
│   │   │   ├── auth.go         # JWT 认证
│   │   │   ├── ratelimit.go    # 限流
│   │   │   ├── validator.go    # 参数校验
│   │   │   ├── metrics.go      # 指标
│   │   │   ├── logger.go       # 日志
│   │   │   ├── error_handler.go # 错误处理
│   │   │   ├── csrf.go         # CSRF 防护
│   │   │   └── security.go     # 安全中间件
│   │   ├── model/              # 数据模型
```

```
│   │   ├── rabbitmq/           # 消息队列
│   │   ├── redis/              # Redis 客户端 + Lua脚本
│   │   ├── registry/           # 服务注册发现
│   │   ├── repository/         # 数据访问层
│   │   ├── response/           # 统一响应 + 错误码
│   │   ├── router/             # 路由配置
│   │   ├── service/            # 业务逻辑
│   │   │   ├── service.go      # 基础服务
│   │   │   ├── seckill.go      # 秒杀核心
│   │   │   └── wechat_pay.go   # 微信支付
│   │   ├── tracing/            # 链路追踪
│   │   ├── circuitbreaker/     # 熔断器
│   │   └── security/           # 安全工具
│   └── scripts/                # 运维脚本
│
├── frontend/                   # 前端应用 (React + TypeScript + Vite)
│   ├── package.json
│   ├── vite.config.ts          # Vite 配置
│   ├── tsconfig.json           # TypeScript 配置
│   └── src/
│       ├── main.tsx            # 应用入口
│       ├── App.tsx             # 主应用组件
│       ├── api/                # API 接口封装
│       │   ├── request.ts      # Axios 实例配置 + 拦截器
│       │   ├── user.ts         # 用户相关 API
│       │   ├── product.ts      # 商品相关 API
│       │   ├── order.ts        # 订单相关 API
│       │   ├── cart.ts         # 购物车 API
│       │   ├── seckill.ts      # 秒杀 API
│       │   └── types.ts        # 类型定义
│       ├── components/         # 可复用组件
│       │   ├── Header.tsx      # 顶部导航栏
│       │   ├── Footer.tsx      # 底部
│       │   ├── ProductCard.tsx # 商品卡片
│       │   └── CartItem.tsx    # 购物车项
│       ├── pages/              # 页面组件
│       │   ├── Home.tsx        # 首页
│       │   ├── Products.tsx    # 商品列表
│       │   ├── ProductDetail.tsx # 商品详情
│       │   ├── Login.tsx       # 登录页
│       │   ├── Register.tsx    # 注册页
│       │   ├── Cart.tsx        # 购物车
│       │   ├── Orders.tsx      # 订单列表
```

```
│       │   ├── Seckill.tsx     # 秒杀活动
│       │   └── Auth.tsx        # 认证页面
│       ├── store/              # 状态管理 (Zustand)
│       │   └── index.ts        # Store 定义
│       └── styles/             # 样式文件
│
├── docs/                       # Swagger API 文档
├── ARCHITECTURE.md             # 架构设计文档
├── README.md                   # 项目说明文档
├── instruction.txt             # 说明文件
└── Makefile                    # 后端 Makefile
```

快速开始

环境要求

组件	版本	说明
Go	1.23+	后端开发
Node.js	18+	前端开发
MySQL	8.0+	数据库
Redis	7.0+	缓存
RabbitMQ	3.12+	消息队列（可选）

1. 克隆项目

```
git clone https://github.com/yourusername/gomall.git
cd gomall
```

2. 启动后端

```
cd backend

# 安装依赖
go mod download
go mod tidy

# 配置数据库
# 编辑 conf/config-dev.yaml 配置 MySQL、Redis、RabbitMQ 连接

# 初始化数据库
mysql -u root -p < deploy/mysql/init.sql

# 启动服务
make run
# 或
go run main.go -env dev
```

3. 启动前端

```
cd frontend

# 安装依赖
npm install

# 启动开发服务器
npm run dev
```

4. 访问服务

服务	地址
前端应用	http://localhost:5173
API 服务	http://localhost:8080
健康检查	http://localhost:8080/health
就绪检查	http://localhost:8080/ready
Prometheus 指标	http://localhost:8080/metrics

服务	地址
Swagger 文档	http://localhost:8080/swagger/index.html

API 文档

用户模块

方法	路径	说明
POST	`/api/user/register`	用户注册
POST	`/api/user/login`	用户登录
GET	`/api/user/profile`	获取用户信息

认证模块

方法	路径	说明
POST	`/api/auth/refresh-token`	刷新Token
POST	`/api/auth/change-password`	修改密码
POST	`/api/auth/logout`	退出登录

商品模块

方法	路径	说明
GET	`/api/product`	商品列表
GET	`/api/product/:id`	商品详情
POST	`/api/product`	创建商品 (需登录)
PUT	`/api/product/:id`	更新商品 (需管理员)
DELETE	`/api/product/:id`	删除商品 (需管理员)

订单模块

方法	路径	说明
POST	`/api/order`	创建订单 (需登录)
GET	`/api/order`	订单列表 (需登录)
GET	`/api/order/:order_no`	订单详情 (需登录)
POST	`/api/order/:order_no/pay`	支付订单 (需登录)
POST	`/api/order/:order_no/cancel`	取消订单 (需登录)

购物车模块

方法	路径	说明
POST	`/api/cart`	添加商品 (需登录)
GET	`/api/cart`	购物车列表 (需登录)
PUT	`/api/cart`	更新数量 (需登录)
DELETE	`/api/cart`	删除商品 (需登录)
DELETE	`/api/cart/clear`	清空购物车 (需登录)

秒杀模块

方法	路径	说明
POST	`/api/seckill`	秒杀接口 (需登录)
POST	`/api/seckill/init`	初始化库存 (需管理员)

微信支付模块

方法	路径	说明
POST	`/api/pay/wechat/unified-order`	统一下单 (需登录)
GET	`/api/pay/wechat/query`	订单查询 (需登录)

方法	路径	说明
POST	`/api/pay/wechat/close`	关闭订单 (需登录)
POST	`/api/pay/wechat/refund`	申请退款 (需登录)
POST	`/api/pay/wechat/notify`	支付回调 (无需认证)

文件上传模块

方法	路径	说明
POST	`/api/upload`	单文件上传 (需登录)
POST	`/api/upload/multi`	多文件上传 (需登录)

统一响应格式

所有 API 返回统一格式：

```
{
  "code": 0,
  "message": "success",
  "data": {...},
  "trace_id": "xxx"
}
```

错误码体系

错误码	说明
0	成功
400	参数错误
401	未登录
403	无权限
404	不存在

错误码	说明
500	系统错误
10001-10099	用户模块
20001-20099	商品模块
30001-30099	订单模块
40001-40099	支付模块
50001-50099	购物车模块
60001-60099	秒杀模块
70001-70099	文件上传模块

核心功能实现

1. 高并发秒杀 (Redis + Lua)

```
// 使用 Lua 脚本原子扣减库存，防止超卖
func decrStockWithLua(ctx context.Context, productID uint, quantity int) (int, error) {
    script := redis.NewScript(`
        local stock = redis.call('GET', KEYS[1])
        if stock == false then return -1 end
        stock = tonumber(stock)
        if stock < tonumber(ARGV[1]) then return -1 end
        redis.call('DECRBY', KEYS[1], ARGV[1])
        return stock - ARGV[1]
    `)
    return script.Run(ctx, redis.Client, []string{key}, quantity).Int()
}
```

秒杀流程：

1. 预加载库存到 Redis
2. 用户请求检查 Redis 库存
3. Lua 脚本原子扣减
4. 发送消息到 RabbitMQ

5. 异步消费者创建订单

2. 统一响应与错误码

```
// 成功响应
response.Ok(c)
response.OkWithData(c, gin.H{"token": token})

// 失败响应
response.BadRequest(c, "参数错误")
response.Unauthorized(c, "未登录")
response.FailWithMsg(c, response.CodeOrderNotFound, "订单不存在")
```

3. JWT 认证与刷新

- bcrypt 密码加密
- JWT Token 生成与验证
- Token 刷新机制（access_token 过期可用 refresh_token 续期）
- 中间件拦截认证

4. 微信支付（沙箱）

```
// 统一下单
result, err := wechatPayService.UnifiedOrder(ctx, orderNo, totalFee, body)

// 支付回调处理
result, err := wechatPayService.PayNotify(ctx, xmlData)
```

5. 限流策略

场景	QPS	突发	实现方式
全局	1000	2000	本地限流
API	100	200	本地限流
秒杀	5	10	Redis 分布式
登录	10	20	本地限流

6. 参数校验

```
type RegisterRequest struct {
    Username string `json:"username" binding:"required,min=3,max=50"`
    Password string `json:"password" binding:"required,min=6,max=20"`
    Email    string `json:"email" binding:"required,email"`
}
```

Docker 部署

使用 Docker Compose

```
# 启动所有服务
docker-compose up -d

# 查看日志
docker-compose logs -f app

# 停止服务
docker-compose down
```

服务端口

服务	端口
GoMall	8080
MySQL	3306
Redis	6379
RabbitMQ	5672 / 15672
Jaeger UI	16686
Jaeger OTLP	4317

Makefile 命令

```
cd backend

make deps           # 下载依赖
make build          # 编译项目
make run            # 开发模式运行
make run-prod       # 生产模式运行
make stop           # 停止服务
make clean          # 清理构建文件
make test           # 运行测试
make docker-build   # 构建 Docker 镜像
make docker-run     # 启动 Docker 服务
make docker-stop    # 停止 Docker 服务
make logs           # 查看日志
make backup         # 数据库备份
make swag           # 生成 Swagger 文档
make help           # 显示帮助
```

项目进度

阶段	状态	内容
Phase 1	✅	单体架构基础、用户/商品/订单模块
Phase 2	✅	高并发秒杀、Redis + Lua + RabbitMQ
Phase 3	✅	购物车模块
Phase 4	✅	可观测性、健康检查、限流、监控
Phase 5	✅	微服务架构、服务注册发现
Phase 6	✅	API规范化、统一响应、错误码体系
Phase 7	✅	JWT刷新、修改密码、退出登录
Phase 8	✅	文件上传功能
Phase 9	✅	微信支付（沙箱）

阶段	状态	内容
Phase 10	✅	前端界面（React + TypeScript）

下一步计划

- [ ] 支付宝支付（正式环境）
- [ ] 订单超时自动取消（定时任务）
- [ ] 消息推送（WebSocket）
- [ ] 缓存优化（多级缓存）
- [ ] 单元测试覆盖率提升
- [ ] Kubernetes 部署支持

贡献指南

1. Fork 本仓库
2. 创建功能分支 (`git checkout -b feature/amazing-feature`)
3. 提交更改 (`git commit -m 'Add amazing feature'`)
4. 推送到分支 (`git push origin feature/amazing-feature`)
5. 创建 Pull Request

许可证

本项目采用 MIT License 开源。

致谢

感谢所有开源贡献者！