Yinli API

基于 Golang Gin 框架构建的高性能 RESTful API 服务，集成了 JWT 认证、Redis 缓存、频率限制、CORS 等安全特性。

🚀 技术架构

核心技术栈

- **Web 框架**: Gin - 高性能的 HTTP Web 框架
- **数据库**: MySQL 8.0 - 关系型数据库
- **缓存**: Redis - 内存数据库，用于缓存和频率限制
- **ORM**: GORM - Go 语言 ORM 库
- **配置管理**: Viper - 配置文件管理
- **认证**: JWT - JSON Web Token 认证
- **文档**: Swagger - API 文档自动生成
- **测试**: Testify - 测试框架
- **容器化**: Docker - 容器化部署

安全特性

- **JWT 认证**: 基于 JSON Web Token 的用户认证
- **CORS 支持**: 跨域资源共享配置
- **频率限制**: 基于 Redis 的 API 频率限制
- **密码加密**: 使用 bcrypt 加密用户密码
- **中间件保护**: 多层中间件安全防护

项目结构

```
yinli-api/
├── cmd/                    # 应用程序入口
│   └── main.go
├── internal/               # 内部应用代码
│   ├── handler/           # HTTP 处理器
│   ├── middleware/        # 中间件
│   ├── model/             # 数据模型
│   ├── repository/        # 数据访问层
│   └── service/           # 业务逻辑层
├── pkg/                    # 可复用的包
│   ├── auth/              # 认证相关
│   ├── cache/             # 缓存操作
│   ├── config/            # 配置管理
│   └── database/          # 数据库连接
├── config/                 # 配置文件
│   ├── dev.yaml           # 开发环境配置
│   ├── stage.yaml         # 预发布环境配置
│   └── prod.yaml          # 生产环境配置
├── docker/                 # Docker 相关文件
├── doc/                    # API 文档
├── sql/                    # 数据库脚本
├── test/                   # 测试文件
├── build/                  # 构建输出
├── Dockerfile              # Docker 镜像构建文件
├── Makefile               # 构建脚本
└── README.md              # 项目说明
```

🛠️ 快速开始

环境要求

- Go 1.21+
- MySQL 8.0+
- Redis 6.0+
- Docker & Docker Compose (可选)

本地开发

1. 克隆项目

```
git clone <repository-url>
cd yinli-api
```

2. 安装依赖

```
make deps
```

3. 配置数据库

```
# 创建数据库
mysql -u root -p < sql/init.sql
```

4. 启动 **Redis**

```
redis-server
```

5. 启动开发环境

```
make dev
```

Docker 部署

1. 生成 **Docker Compose** 文件

```
make docker-compose-dev
```

2. 启动服务

```
make docker-up-dev
```

📋 可用命令

开发命令

```
make dev          # 启动开发环境
make stage        # 启动预发布环境
make prod         # 启动生产环境
```

构建命令

```
make build        # 构建 Linux 版本
make build-all    # 构建所有平台版本
make clean        # 清理构建文件
```

测试命令

```
make test         # 运行测试
make test-coverage # 运行测试并生成覆盖率报告
make benchmark    # 运行基准测试
```

代码质量

```
make fmt          # 格式化代码
make vet          # 静态检查
make lint         # 代码规范检查
make check        # 执行所有检查
```

文档生成

```
make docs         # 生成 API 文档
make docs-serve   # 启动文档服务器
```

Docker 操作

```
make docker-build      # 构建 Docker 镜像
make docker-up-dev     # 启动开发环境容器
make docker-up-stage   # 启动预发布环境容器
make docker-up-prod    # 启动生产环境容器
make docker-down       # 停止容器
make docker-logs       # 查看容器日志
```

🔧 配置说明

环境配置

项目支持三种环境配置：

- `dev` - 开发环境
- `stage` - 预发布环境
- `prod` - 生产环境

配置文件位于 `config/` 目录下，可通过环境变量 `APP_ENV` 或命令行参数 `-env` 指定。

主要配置项

```yaml
server:
  port: 1234              # 服务端口
  mode: debug             # 运行模式 (debug/release)

database:
  host: localhost         # 数据库地址
  port: 3306              # 数据库端口
  username: root          # 数据库用户名
  password: sasasasa      # 数据库密码
  dbname: yinli           # 数据库名称

redis:
  host: localhost         # Redis 地址
  port: 6379              # Redis 端口
  password: ""            # Redis 密码
  db: 0                   # Redis 数据库

jwt:
  secret: your-secret-key # JWT 密钥
  expireHours: 24         # 令牌过期时间(小时)

rateLimit:
  enabled: true           # 是否启用频率限制
  requests: 100           # 每个时间窗口的请求数
  window: 60              # 时间窗口(秒)
```

📡 API 接口

认证接口

方法	路径	描述	认证
POST	`/api/auth/register`	用户注册	✖
POST	`/api/auth/login`	用户登录	✖

用户接口

方法	路径	描述	认证
GET	`/api/user/profile`	获取用户资料	✅
PUT	`/api/user/profile`	更新用户资料	✅
PUT	`/api/user/password`	修改密码	✅

管理员接口

方法	路径	描述	认证
GET	`/api/admin/users`	获取用户列表	✅ (管理员)
DELETE	`/api/admin/users/{id}`	删除用户	✅ (管理员)
PUT	`/api/admin/users/{id}/status`	更新用户状态	✅ (管理员)

系统接口

方法	路径	描述	认证
GET	`/health`	健康检查	❌
GET	`/swagger/*`	API 文档	❌

🧪 接口测试

使用 curl 测试

1. 用户注册

```
curl -X POST http://localhost:1234/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{
    "name": "testuser",
    "password": "password123"
  }'
```

2. 用户登录

```
curl -X POST http://localhost:1234/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "name": "testuser",
    "password": "password123"
  }'
```

3. 获取用户资料

```
curl -X GET http://localhost:1234/api/user/profile \
  -H "Authorization: Bearer YOUR_JWT_TOKEN"
```

使用 Postman

1. 导入 Postman 集合文件（如果有）
2. 设置环境变量 `base_url` 为 `http://localhost:1234`
3. 在认证接口获取 JWT token
4. 在需要认证的接口中添加 `Authorization: Bearer {token}` 头

🔍 开发调试

日志查看

```
# 查看应用日志
tail -f logs/app.log

# 查看 Docker 容器日志
make docker-logs
```

数据库调试

```
# 连接数据库
mysql -h localhost -u root -p yinli

# 查看用户表
SELECT * FROM user;
```

Redis 调试

```
# 连接 Redis
redis-cli

# 查看所有键
KEYS *

# 查看频率限制
KEYS rate_limit:*
```

性能监控

```
# 查看应用性能
go tool pprof http://localhost:1234/debug/pprof/profile

# 内存使用情况
go tool pprof http://localhost:1234/debug/pprof/heap
```

📊 测试报告

运行测试并生成报告：

```
make test-coverage
```

测试报告将生成在 `build/coverage.html`，可在浏览器中查看详细的覆盖率信息。

🚀 部署指南

生产环境部署

1. 构建生产版本

```
make build
```

2. 配置生产环境

```
# 修改 config/prod.yaml
# 设置正确的数据库和 Redis 连接信息
# 更改 JWT 密钥
```

3. 使用 **Docker** 部署

```
make docker-compose-prod
make docker-up-prod
```

环境变量

生产环境建议使用环境变量覆盖敏感配置：

```
export YINLI_DATABASE_PASSWORD=your_db_password
export YINLI_JWT_SECRET=your_jwt_secret
export YINLI_REDIS_PASSWORD=your_redis_password
```

📚 主要依赖库

核心依赖

- Gin Web Framework - HTTP Web 框架
- GORM - ORM 库
- Viper - 配置管理
- JWT-Go - JWT 实现
- Go-Redis - Redis 客户端

中间件

- Gin-CORS - CORS 中间件
- Gin-Swagger - Swagger 文档

测试工具

- Testify - 测试断言库
- HTTP Test - HTTP 测试工具

开发工具

- Air - 热重载工具
- GolangCI-Lint - 代码检查工具

🤝 贡献指南

1. Fork 项目
2. 创建特性分支 (`git checkout -b feature/AmazingFeature`)
3. 提交更改 (`git commit -m 'Add some AmazingFeature'`)
4. 推送到分支 (`git push origin feature/AmazingFeature`)
5. 开启 Pull Request

📄 许可证

本项目采用 MIT 许可证 - 查看 LICENSE 文件了解详情。

🆘 常见问题

Q: 如何修改数据库连接？

A: 修改 `config/{env}.yaml` 文件中的 `database` 配置项。

Q: 如何添加新的 API 接口？

A: 1. 在 `internal/handler` 中添加处理函数
2. 在 `internal/handler/router.go` 中注册路由
3. 添加相应的测试用例

Q: 如何自定义中间件？

A: 在 `internal/middleware` 目录下创建新的中间件文件，参考现有中间件的实现。

Q: 如何部署到生产环境？

A: 使用 `make docker-compose-prod` 生成生产环境配置，然后使用 `make docker-up-prod` 部署。