

# 目 录

介绍

控制台进度条

**NutsDB**

微软Go教程

批量重命名工具

**Dapr**

滴滴开源项目

**Go 泛型**

国人开源监控

**SQL工具**

**sqlx**

**Go 表单验证器**

眼睛随鼠标移动

**fiber浏览器**

前端转Go喜欢的框架

**Go 分布式文件系统**

图片服务器

**Go直播项目**

## 介绍

### 版权声明

本文档转自整理:[转角遇到GitHub](#)

发现 GitHub 好项目，开源爱好者！包括但不限于 Python、Java、Go、Rust、JavaScript、AI 等技术，开源学习和资讯。

## 控制台进度条

大家好，我是欧盆索思（opensource），每天为你带来优秀的开源项目！

据 2020 年 Go 官方调查报告显示，使用 Go 进行 CLI 开发排名第二。Go 爱好者们，应该也有不少用 Go 写命令行程序的。

今天推荐一个命令行程序有用的辅助库：控制台进度条。

项目地址：<https://github.com/cheggaaa/pb>，Star 数：2.7k+。

看一个简单的使用例子：

```
package main

import (
    "time"

    "github.com/cheggaaa/pb/v3"
)

func main() {
    count := 100000
    // create and start new bar
    bar := pb.StartNew(count)

    // start bar from 'default' template
    // bar := pb.Default.Start(count)

    // start bar from 'simple' template
    // bar := pb.Simple.Start(count)

    // start bar from 'full' template
    // bar := pb.Full.Start(count)

    for i := 0; i < count; i++ {
        bar.Increment()
        time.Sleep(time.Millisecond)
    }
    bar.Finish()
}
```

运行结果类似这样：

```
> go run test.go
37158 / 100000 [=====>] 37.16% 1m11s
```

如果不喜欢这个简单的样式，可以自己进行简单的定制。

目前该库最新版本是 v3，因此建议这么使用（基于 Module）：

```
go get github.com/cheggaaa/pb/v3
```

# NutsDB

大家好，我是欧盆索思（opensource），每天为你带来优秀的开源项目！

## 一句话描述

NutsDB 是纯 Go 语言编写一个简单、高性能、内嵌型、持久化的 key-value 数据库。

项目地址：<https://github.com/xujiajun/nutsdb>，Star 数：1.5k+。

## 简介

NutsDB 支持事务，从 v0.2.0 之后的版本开始支持 ACID 的特性，建议使用最新的 release 版本。v0.2.0 之前的版本，保持高性能，没有作 sync，但是具备高性能的写（本地测试，百万数据写入达 40~50W+/s）。所有的操作都在事务中执行。NutsDB 从 v0.2.0 版本开始支持多种数据结构，如列表(list)、集合(set)、有序集合(sorted set)。从 0.4.0 版本开始增加自定义配置读写方式、启动时候的文件载入方式、sync 是否开启等。

### 为什么有 NutsDB?

想找一个用纯 go 编写，尽量简单（方便二次开发、研究）、高性能（读写都能快一点）、内嵌型的（减少网络开销）数据库，最好支持事务。因为我觉得对于数据库而言，数据完整性很重要。如果能像 Redis 一样支持多种数据结构就更好了。而像 Redis 一般用作缓存，对于事务支持也很弱。此外，对于如何实现 kv 数据库的好奇心吧。数据库可以说是系统的核心，了解数据库的内核或者自己有实现，对更好的用轮子或者下次根据业务定制轮子都很有帮助。天下没有银弹，NutsDB 也有他的局限，比如随着数据量的增大，索引变大，启动会慢，只想说 NutsDB 还有很多优化和提高的空间，由于本人精力以及能力有限。所以把这个项目开源出来。更重要的是我认为一个项目需要有人去使用，有人提意见才会成长。

## Example

官方提供了各种操作的例子代码，见：<https://github.com/xujiajun/nutsdb/tree/master/examples>。

## Doc

该项目是国人主导开发的，有中文文档：<https://github.com/xujiajun/nutsdb/blob/master/README-CN.md>。

## QA

1、Q: 现有的不能满足需求吗？  
A: 对比了如下几个：

### BoltDB

BoltDB 和 NutsDB 很相似都是内嵌型的 key-value 数据库，同时支持事务。Bolt 基于 B+tree 引擎模型，只有一个文件，NutsDB 基于 bitcask 引擎模型，会生成多个文件。当然他们都支持范围扫描和前缀扫描这两个实用的特性。

### LevelDB, RocksDB

LevelDB 和 RocksDB 都是基于 LSM tree 模型。不支持 bucket。其中 RocksDB 目前还没看到 golang 实现的版本。

### Badger

Badger 也是基于 LSM tree 模型。但是写性能没有我想象中高。不支持 bucket。  
另外，以上数据库均不支持多种数据结构如 list、set、sorted set，而 NutsDB 从 0.2.0 版本开始支持这些数据结构。

## 微软Go教程

在微软 Docs 网站发现了一个 Go 教程，英文叫做：《Take your first steps with Go》，教程是英文的，但微软的国际化支持，应该很快就会有中文版本。而且现在介绍页已经有中文的了。

教程地址：<https://docs.microsoft.com/zh-cn/learn/paths/go-first-steps/>。

根据该教程的建议，学习完大概需要 5 小时 24 分钟，这是一个入门教程。主要包括如下内容：

- 安装编写第一行 Go 代码所需的工具。
- 了解如何在 Go 中使用控制流。
- 了解 Go 的数据类型。
- 了解如何处理错误。
- 使用方法和接口
- 了解 Go 中并发的工作原理。
- 编写并测试程序

大概浏览了下这个教程，其中涉及了较多使用示例，每个模块会有学习目标、学习之前需要具备的条件，学习完每一个模块会有测试题，方便验证学习成果，还支持在线提交答案。

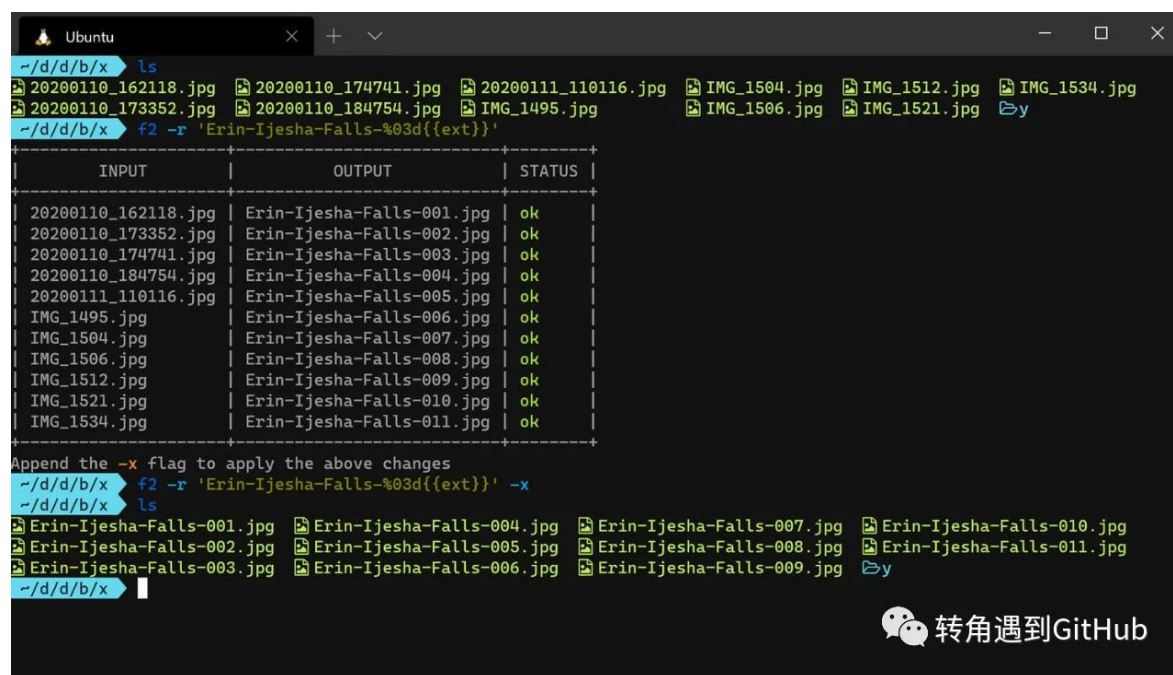
此外，每个模块结尾会对该模板进行知识点总结，帮你进行整理。  
希望这个教程对大家学习 Go 语言有帮助！坐等中文版。

## 批量重命名工具

大家好，我是欧盆索思（opensource），每天为你带来优秀的开源项目！

F2 是一个快捷键，进行文件重命名。今天推荐一个开源工具，它的名字就叫做 f2，是一个跨平台、快速、安全的批量重命名工具。

项目地址：<https://github.com/ayoisaiiah/f2>，刚开源，使用 Go 语言实现的。



转角遇到GitHub

它核心的特色是安全和快速。在速度方面，对比了几个工具：

- rnm[1] (C++) — v4.0.9
- rnr[2] (Rust) — v0.3.0
- brename[3] (Go) — v2.11.0

f2 的结论是它比以上工具都快 2~3 倍。

具体其他特色有：

- 跨平台，支持 Linux、MacOS 和 Windows；
- 支持正则表达式过滤文件，并支持分组；
- 安全，体现在默认执行改名操作，而是预览，只有告诉它改名它才执行；
- 支持递归重命名；
- 支持撤回；
- . . . .

安装该工具，可以通过 go get 安装，也可以通过 npm 安装，因为发布到了 npm 上：

```
$ npm i @ayoisaiiah/f2 -g
```

以下是 f2 的命名帮助文档：

```

DESCRIPTION:
  f2 is a command-line tool for batch renaming multiple files and directories quickly and safely

USAGE:
  f2 FLAGS [OPTIONS] [PATHS...]

AUTHOR:
  Ayooluwa Isaiah <ayo@freshman.tech>

VERSION:
  v1.1.0

FLAGS:
  --find string, -f string  Search string or regular expression.
  --replace string, -r string  Replacement string. If omitted, defaults to an empty string.
  --start-num value, -n value  Starting number when using numbering scheme in replacement string such as %
03d (default: 1)
  --output-file value, -o value  Output a map file for the current operation
  --exec, -x  Execute the batch renaming operation (default: false)
  --recursive, -R  Rename files recursively (default: false)
  --undo value, -u value  Undo a successful operation using a previously created map file
  --ignore-case, -i  Ignore case (default: false)
  --ignore-ext, -e  Ignore extension (default: false)
  --include-dir, -d  Include directories (default: false)
  --only-dir, -D  Rename only directories (implies include-dir) (default: false)
  --hidden, -H  Include hidden files and directories (default: false)
  --fix-conflicts, -F  Fix any detected conflicts with auto indexing (default: false)
  --help, -h  show help (default: false)
  --version, -v  print the version (default: false)

WEBSITE:
  https://github.com/ayoisaiiah/f2

```

看一个具体简单的例子:

```

$ f2 -f 'Screenshot' -r 'Image'
+-----+-----+-----+
| INPUT | OUTPUT | STATUS |
+-----+-----+-----+
| Screenshot (1).png | Image (1).png | ok |
| Screenshot (2).png | Image (2).png | ok |
| Screenshot (3).png | Image (3).png | ok |
+-----+-----+-----+

```

当不提供 `-x` Flag 时, 只是展示改名后的样子, 只有加上了 `-x`, 才会真正的执行改名操作。在项目主页提供了其他例子供参考。

此外, 如果你是一个 Go 爱好者, 可以学习下它的源码, 它使用了 [github.com/urfave/cli](https://github.com/urfave/cli) 这个库, 我也是很喜欢这个库。学习开源项目源码, 对自己能力会很有帮助。

#### 参考资料

- [1] rnm: <https://github.com/neurobin/rnm>
- [2] rnr: <https://github.com/ChuckDaniels87/rnr>
- [3] brename: <https://github.com/shenwei356/brename>

# Dapr

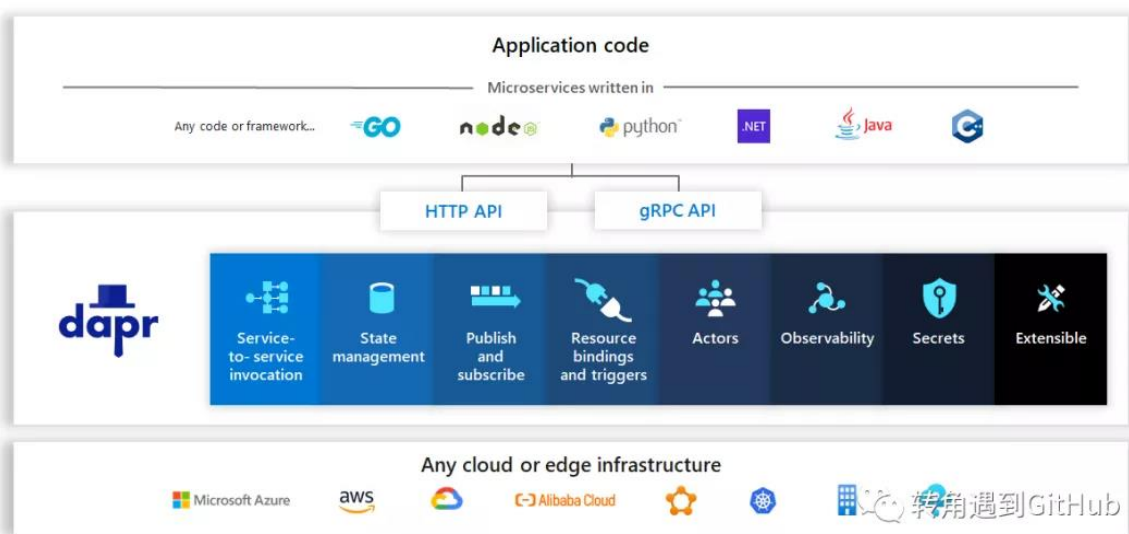
大家好，我是欧益索思（opensource），每天为你带来优秀的开源项目！

微软真是越来越棒，不断拥抱开源，拥抱外界新技术，比如 Go、Rust 等。今天推荐的这个项目是一个 Go 语言项目。

项目名：Dapr，地址：<https://github.com/dapr/dapr>，Star 数：7.4k+。

Dapr 是一种可移植的，无服务器的，事件驱动的运行，它使开发人员可以轻松跨云和边缘构建并运行包含多种语言和开发人员框架的弹性、无状态或有状态微服务。它的口号是：任何语言，任何框架，任何地方。

Dapr 将微服务应用程序构建为开放的，独立的构建块的最佳实践进行了整理，使你能够使用自己喜爱的语言和框架来构建可移植的应用程序。每个构建块都是独立的，你可以在应用程序中使用其中的一个，部分或全部。



## 项目目标

- 使开发人员可以使用任何语言或框架来编写分布式应用程序
- 通过提供最佳实践构建块来解决开发人员构建微服务应用程序时遇到的难题
- 社区驱动，开放并与供应商保持中立
- 通过开放的API提供一致性和可移植性
- 跨云和边缘，与平台无关
- 拥抱可扩展性并提供可插入组件，而无需锁定供应商
- 通过高性能和轻量级实现物联网和边缘计算场景
- 可以从现有代码中逐步采用，而没有运行时依赖

## 为什么要有 Dapr?

编写高性能，可伸缩和可靠的分布式应用程序很困难。Dapr 为你带来了行之有效的模式和实践。它将事件驱动和参与者的语义统一到一个简单，一致的编程模型中。它支持不需要特定框架的所有编程语言。您不会遇到低级原语，例如线程，并发控制，分区和缩放。相反，您可以使用所选的熟悉的 Web 框架通过实现简单的 Web 服务器来编写代码。

Dapr 在线程和状态一致性模型方面很灵活。如果愿意，可以利用多线程，还可以在不同的一致性模型中进行选择。这种灵活性使得无需人工约束即可实施高级方案。Dapr 是独一无二的，因为您可以在平台和基础实现之间无缝过渡，而无需重写代码。

另外，如果你对该项目感兴趣，除了学习官方文档，还可以跟着这份 Demo 学习。<https://github.com/mchmarny/dapr-demos>。



## Dapr

注意，目前该项目还存于 alpha 状态，不建议用于生产环境。

# 滴滴开源项目

作为卓越的一站式移动出行和生活平台，滴滴在亚洲、拉美和澳洲为超过5.5亿用户提供出租车、快车、专车、豪华车、公交、代驾、企业级、共享单车、共享电单车、汽车服务、外卖、支付等多元化的服务。滴滴平台上，有数千万车主及司机获得灵活的工作和收入机会，年运送乘客超过100亿人次。

滴滴开源的三年时间里，在滴滴高级副总裁、开源委员会主席章文嵩博士的倡导下，秉持「拥抱开放、合作共赢、创造价值」的理念。滴滴积极参与业界的开源项目，并不断对外输出内部的优秀项目。自2017年6月30日，滴滴首个开源项目VirtualAPK发布起，滴滴已发布了40+个项目，获得了 6.3W star，在社区获得了良好反响。以下小编将带大家速览滴滴的40+开源项目。（按首字母先后排序）

## 1.人工智能

### ▮ AoE

AoE 取自AI on Edge，是一个终端侧AI集成运行时环境(IRE)。AoE 以“稳定性、易用性、安全性”为设计原则，帮助开发者将不同框架的深度学习算法轻松部署到终端高效执行。

项目地址：<https://github.com/didi/aoe>

### ▮ Athena

Athena是端到端自动语音识别（ASR）引擎的开源实现。目前，该项目支持基于连接器时间分类（CTC）的模型，基于变压器的编解码器模型和基于混合CTC/注意力的模型以及基于MPC的无监督预转换的训练和解码。我们的愿景是增强语音识别的端到端模型的工业应用和学术研究。为了使所有人都能使用ASR，我们还发布了一些示例示例，这些示例实现基于一些开源数据集，例如HKSUT，Librispeech我们所有的模型都在Tensorflow> = 2.0.0中实现。

项目地址：<https://github.com/didi/athena>

### ▮ Chinese NLP

中文自然语言处理（NLP）的相关链接：[数据集和最新结果](#)。

项目地址：<https://github.com/didi/ChineseNLP>

### ▮ Delta

DELTA是一个基于深度学习的语音和自然语言理解模型平台，旨在打造一个便捷使用、简洁上线、快捷开发的工业级的语音和自然语言理解的模型框架。

项目地址：<https://github.com/didi/delta>

### ▮ DLFlow

DLFlow是一套深度学习pipeline，它结合了Spark的大规模特征处理能力和Tensorflow模型构建能力。利用DLFlow可以快速处理原始特征、训练模型并进行大规模分布式预测，十分适合离线环境下的生产任务。

项目地址：<https://github.com/didi/dlflow>

### ▮ HetSANN

AAAI'20论文中的HetSANN的源代码：异构结构学习的基于注意力的图神经网络。

项目地址：<https://github.com/didi/hetsann>

### ▮ maskdetection

为了进一步帮助抵抗冠状病毒，滴滴出行决定向公众免费开放其面罩检测技术。由DiDi AI团队开发的遮罩检测技术基于DFS面部检测算法，DiDi在其平台上采用了面部属性识别算法。该模型克服了一些困难，例如白天复杂的光照变化，面部姿势变化，面部比例等。它使用加权损失函数和数据增强方法来处理白天和晚上的不同蒙版类型和不均匀蒙版数据。

该系统可以使用上传的图像识别非掩膜驾驶员，其准确度为99.5%，并且在DiDi车载摄像头的现场检查中，可以达到98%的准确度。该模型在200,000张面孔的数据集上进行了训练，以确保其鲁棒性。

这种快速检测系统可广泛用于旅行场景中，包括手机照片，监视图像等，并且能够全天候工作。

项目地址：<https://github.com/didi/maskdetection>

### ▮ SQLFlow

滴滴DS统计科学团队立足于公司内部常用业务场景的需求出发，与蚂蚁金服SQLFlow初始研发团队之间从模型、社区、与文化上全方位合作共建。双方能力优势互补，资源协同性强，旨在推进并实现只要懂商业逻辑就能用上人工智能，让最懂业务的人也能无障碍地使用人工智能。

项目地址：<https://github.com/sql-machine-learning/sqlflow>

## 2. 智慧交通

### ALITA/ALITA-UI

ALITA (A Layered Instrument To Analyze) 是由小桔车服数据部出品的一种基于图层展示数据分析工具, 我们通过大量业务需求实现沉淀抽象出通用化的“点”、“线”、“面”三种数据模型, 和地图丰富的点、线、面设计元素完美融合, 设计并封装前后端通用的模块化组件, 实现快速搭建地图应用。

项目地址: <https://github.com/didi/ALITA>

### mtmc-vt

该代码用于AI City Challenge 2019 Track1, MTMC Vehicle Tracking。

项目地址: <https://github.com/didi/mtmc-vt>

### TrafficIndex

TTI是业内使用较多的城市拥堵程度的评价指标, 是反应实际花费的行程时间与自由流花费行程时间的比值关系, 值越大表示交通运行状态越差, 一般与拥堵程度正相关, 其它如异常天气(如雨、雪、雾等)或者异常道路情况也可能对TTI的数值产生影响。

项目地址: <https://github.com/didi/TrafficIndex>

## 3. 中间件与架构

### Booster

Booster 是一款专门为移动应用设计的易用、轻量级且可扩展的质量优化框架, 通过动态发现和加载机制提供可扩展的能力。它不仅仅是一个框架, 更内置了丰富的质量优化工具。

项目地址: <https://github.com/didi/booster>

### DDMQ

DDMQ 是滴滴出行架构部基于 Apache RocketMQ 构建的消息队列产品。作为分布式消息中间件, DDMQ 为滴滴出行各个业务线提供了低延迟、高并发、高可用、高可靠的消息服务。DDMQ 提供了包括实时消息、延迟消息和事务消息在内的多种消息类型以满足不同的业务需求。用户通过统一的 Web 控制台和傻瓜式的 SDK 即可轻松接入 DDMQ 生产和消费消息, 体验功能丰富、稳定的消息服务。

项目地址: <https://github.com/didi/DDMQ>

### Go-Spring

Go-Spring 的愿景是让 GoLang 程序员也能用上如 Java Spring 那般威力强大的编程框架。特性: 提供完善的 IoC 容器, 支持依赖注入、属性绑定; 提供强大的启动器框架, 支持自动装配、开箱即用; 提供常见组件的抽象层, 支持灵活地替换底层实现; Go-Spring 当前使用 Go1.12 进行开发, 使用 Go Modules 进行依赖管理。

项目地址: <https://github.com/didi/go-spring>

## 4. 大数据

### ES-Fastloader

ES-Fastloader是一种能够快速地为海量离线数据生成索引的方案, 采用了一种Build & Push的机制, 利用Hadoop强大的并行计算能力, 可在1-2小时内极快速构建几十TB级的数据, 解决海量数据构建ES时, 索引文件构建时效低的问题。

项目地址: <https://github.com/didi/ES-Fastloader>

### Levin

Levin是针对低频更新、静态使用、大规模数据的快速加载解决方案。Levin实现了一套使用在shm共享内存片段上的STL-like容器, 高效托管大规模静态数据, 加速大内存服务冷启动和热加载。具备简单易用、效率高、性能好、内存省的优点。

项目地址: <https://github.com/didi/levin>

## 5. 运维监控

### Elastic-trib

Elastic-trib是一个Elasticsearch集群管理命令行工具, 用于管理公司内部30多个Elasticsearch集群, 方便用于集成到shell脚本及通过命令行终端对集群进行管控。

项目地址: <https://github.com/didi/elastic-trib>

### falcon-log-agent

falcon-log-agent是一个开源版的日志采集工具, 旨在从流式的日志中抓取、统计日志中的特征信息。获取的特征信息, 与开

源版Open-Falcon监控系统打通。可用于业务指标的衡量、也可用于稳定性的建设。

项目地址: <https://github.com/didi/falcon-log-agent>

#### | Kafka-Manager

一站式Apache Kafka集群指标监控与运维管控平台。

项目地址: <https://github.com/didi/kafka-manager>

#### | Nightingale

夜莺(Nightingale)是滴滴基础平台联合滴滴云研发和开源的企业级监控解决方案。旨在满足云原生时代企业级的监控需求。Nightingale在产品完成度、系统高可用、以及用户体验方面,达到了企业级的要求,可满足不同规模用户的场景,小到几台机器,大到数十万都可以完美支撑。兼顾云原生和裸金属,支持应用监控和系统监控,插件机制灵活,插件丰富完善,具有高度的灵活性和可扩展性。

项目地址: <https://github.com/didi/nightingale>

#### | sqt

此进程在滴滴云上用于管理机器上面的其他agent,比如监控的agent、安全的agent,管理主要是:安装、升级、卸载、查看启动状态,不做其他事情。省去客户手工安装其他agent的工作。

项目地址: <https://github.com/didi/sqt>

## 6. 小程序

#### | Chameleon

cml作为真正让一套代码运行多端的框架,提供标准的MVVM模式,统一开发各类终端。同时,拥有各端独立的运行时框架(runtime)、数据管理(store)、组件库(ui)、接口(api)。此外,cml在跨端能力加强、能力统一、表现一致等方面做了许多工作。

项目地址: <https://github.com/didi/chameleon>

#### | Mpx

Mpx是一款致力于提高小程序开发体验和开发效率的增强型小程序框架,通过Mpx,我们能够高效优雅地开发出具有极致性能的优质小程序应用,并将其输出到各大小程序平台和web平台中运行。

项目地址: <https://github.com/didi/mpx>

## 7. 移动开发

#### | Echo

Echo是一款简单易用、插件化易扩展、大屏显示和操作的调试工具,旨在提高移动端的研发调试效率。基于现有的一些问题和团队需求,我们开发了一款桌面端的调试工具Echo,它可以帮助我们实时查看App各类数据(网络请求、日志、埋点等),也可以无须改动代码快速修改调试App的UI效果,提高我们的开发调试效率。

项目地址: <https://github.com/didi/echo>

#### | DoraemonKit

简称DoKit,中文名哆啦A梦,意味着能够像哆啦A梦一样提供给他的主人各种各样的工具。是一款功能齐全的客户端(iOS、Android)研发助手。能够让每一个App快速接入一些常用的或者你没有实现的一些辅助开发工具、测试效率工具、视觉辅助工具,而且能够完美在Doraemon面板中接入你已经实现的与业务紧密耦合的一些非通用的辅助工具,并搭配我们的dokit平台,让功能得到延伸,接入方便,便于扩展。

项目地址: <https://github.com/didi/DoraemonKit>

#### | DroidAssist

DroidAssist是一个轻量级的Android字节码编辑插件,通过在xml进行简单的配置即可实现对class文件进行动态修改。

项目地址: <https://github.com/didichuxing/DroidAssist>

#### | VirtualAPK

VirtualAPK是滴滴出行自研的一款优秀的插件化框架,通过将业务模块插件化,可随时更新插件来发布新功能,具备版本随时发布的能力。该款插件化框架可帮助企业随时通过更新插件的方式来发布新功能,包括修复严重Crash或进行业务“试错”,拥有功能完备、基本无入侵、插件可轻松访问宿主代码和资源、高兼容性这四大核心优势。

项目地址: <https://github.com/didi/VirtualAPK>

## 8. 系统工具

### Collection

Collection包目标是用于替换golang原生的Slice，使用场景是在大量不追求极致性能，追求业务开发效能的场景。Collection包目前支持的元素类型：int, int64, float32, float64, string, struct。

项目地址：<https://github.com/didi/collection>

### GateKeeper

GateKeeper 是一个使用 Go (golang) 编写的不依赖分布式数据库的 API 网关，使用它可以高效进行服务代理 以及 在线化服务配置并且你无需重启服务器。

项目地址：<https://github.com/didi/gatekeeper>

### Gendry

Gendry是一个用于辅助操作数据库的Go包。基于go-sql-driver/mysql，它提供了一系列的方法来为你调用标准库database/sql中的方法准备参数。

项目地址：<https://github.com/didi/gendry>

### JuShaTa

JuShaTa是一个Java容器，提供模块隔离及模块热加载能力。我们提供了一个类似于Tomcat的Java容器JuShaTa，在JuShaTa容器中每个SpringBoot服务都是一个独立的模块。通过自定义ClassLoader，不同模块使用不同的ClassLoader进行加载，解决jar包冲突；使用Spring Context进行上下文隔离，每个模块对应一个Context，解决bean冲突。

项目地址：<https://github.com/didi/JuShaTa>

### Kemon

一个用于macOS内核监视的基于开源前后回调的框架。kemon是一个基于macOS的内核监控框架，在Windows上内核监控有很好的基础，但在Mac上还没有一个较成熟的hook框架，去跟踪更多的进程内核事件的发生。Kemon进一步完善，为macOS的内核安全性的提高作出应有的贡献。

项目地址：<https://github.com/didi/kemon>

### SDS

SDS（即 Service Downgrade System）是一个轻量级、简单、易用的限流、熔断、降级系统，能让Java应用做到自动限流、熔断和快速恢复，提升应用整体的“弹性”。现在服务端通过采用流行的微服务架构来应对错综复杂的大流量场景，并在业务高速发展时仍然能做到较强的快速迭代能力和可扩展性。微服务架构并不是将整个系统变得更简单，相反，微服务架构的管理难度高于普通的集中式架构，所以，如何保证系统的每个节点在错综复杂的环境下能稳定提供服务，需要借助工具来让服务节点能抵挡流量冲击、熔断依赖坏点。

项目地址：<https://github.com/didi/sds>

### Tinyid

Tinyid是用Java开发的一款分布式id生成系统，基于数据库号段算法实现，关于这个算法可以参考美团leaf或者tinyid原理介绍。Tinyid扩展了leaf-segment算法，支持了多db(master)，同时提供了java-client(sdk)使id生成本地化，获得了更好的性能与可用性。Tinyid在滴滴客服部门使用，均通过tinyid-client方式接入，每天生成亿级别的id。

项目地址：<https://github.com/didi/tinyid>

## 9. 前端

### cube-ui

应用Vue构建的出色移动端ui库工具。该技术拥有质量可靠、体验极致、标准规范和强扩展性这四大特点，并拥有独特的后编译技术方案帮助大幅优化性能。cube-ui 的目标是让移动端的开发更容易，让开发人员更加专注于业务逻辑的开发，提升研发效率。

项目地址：<https://github.com/didi/cube-ui>

### di18n

di18n 是一个自动转换、基于配置的前端国际化方案。它能自动扫描代码中的中文文案，将其替换成国际化标记；同时将语言抽取成配置，可以放到服务端保存及更新。

项目地址：<https://github.com/didi/di18n>

### Epage

Epage是一款基于schema的可视化页面配置工具。由工单系统流程表单场景抽象而来，升级成为支持跨框架、组件库渲染、可定制的通用页面配置工具。常用于流程表单、中后台页面配置。

项目地址：<https://github.com/didi/epage>

#### ■ Mand Mobile

Mand Mobile提供了30+的实用组件，能够满足移动端页面开发中的大部分需求。其中的业务类组件还针对金融领域，提取了包括图表、数字键盘等，从而更好地满足相关产品的开发需要。

项目地址：<https://github.com/didi/mand-mobile>

#### ■ Mand Mobile-RN

Mand Mobile RN 是 Mand Mobile for React Native 的简称。作为 Mand Mobile 系列的React Native 组件库，在聚焦金融场景的同时，还提供了一些在 RN 项目中特有的组件和解决方案，皆在提升跨端项目的开发效率和UI体验。

项目地址：<https://github.com/didi/mand-mobile-rn>

#### ■ Pile.js

使用React构建的轻量级移动组件库。设计、前端高效协同，快速搭建移动端组件库。

项目地址：<https://github.com/didi/pile.js>

#### ■ Tips

Tips是一个静态文案管理平台。用于修改Web页面的静态文案，支持文案国际化，并提供提示信息的UI展示。它的目的是解决前端开发者频繁的静态文案修改问题，避免因为简单的文案修改而发起复杂的上线流程。

项目地址：<https://github.com/didi/Tips>

## 10. 研发测试

#### ■ benchmark-thrift

benchmark-thrift是一款测试Thrift应用程序性能的工具，开箱即用，高效简单。

项目地址：<https://github.com/didi/benchmark-thrift>

#### ■ Rdebug

Rdebug 是滴滴开源的一款用于 RD 研发、自测、调试的实用工具，可以被用来提升 RD 研发效率、保障代码质量进而减少线上事故。

项目地址：<https://github.com/didi/rdebug>

#### ■ Sharingan

Sharingan是一个基于golang的流量录制回放工具，录制线上真实请求流量进行回放测试，适合项目重构、回归测试等。随着微服务架构的兴起，服务之间的依赖关系变的越来越复杂，系统升级频繁导致维护测试用例成本高，依赖下游众多也很难提供稳定的测试环境，为此，我们开发这套工具来缓解上述问题。

项目地址：<https://github.com/didi/sharingan>

#### ■ thrift-mock

thrift-mock是一款轻量级的Java测试工具，用来模拟thrift服务。通过它可以轻松的将依赖的thrift服务接口进行mock，获得指定的接口返回，从而极大的提升了联调、测试阶段的开发效率。

项目地址：<https://github.com/didi/thrift-mock>

## Go 泛型

泛型一直以来是基于 `dev.typeparams` 分支开发的，而每次版本的功能开发是在 `master` 进行的，在要发布时基于 `master` 打出具体的版本分支。

前段时间，泛型已经被正式接受，经过讨论，泛型的开发计划在 `master` 进行，最近已经将泛型分支合入 `master` 了，具体见：（阅读原文可以直达该链接）

<https://github.com/golang/go/commit/9a99515c8f1febe39151a80b2088d0d6fdc55ca2>

这表明，`Go1.17` 中将包含泛型的代码，但 `Go1.17` 是否可以试用泛型，目前不知晓，要看 `Go` 怎么处理。原计划泛型是在 `Go1.18` 发布的。期待~

# 国人开源监控

## 项目简介

集监控点监控、日志监控、数据可视化以及监控告警为一体的国产开源监控系统，直接部署即可使用。



监控数据类型丰富，提供多种富有表现力的图表，满足对数据可视化的需要，目前支持折线图、饼图、地理位置图，后续会引入更多富有表现力的图表以加强对数据可视化的支持。





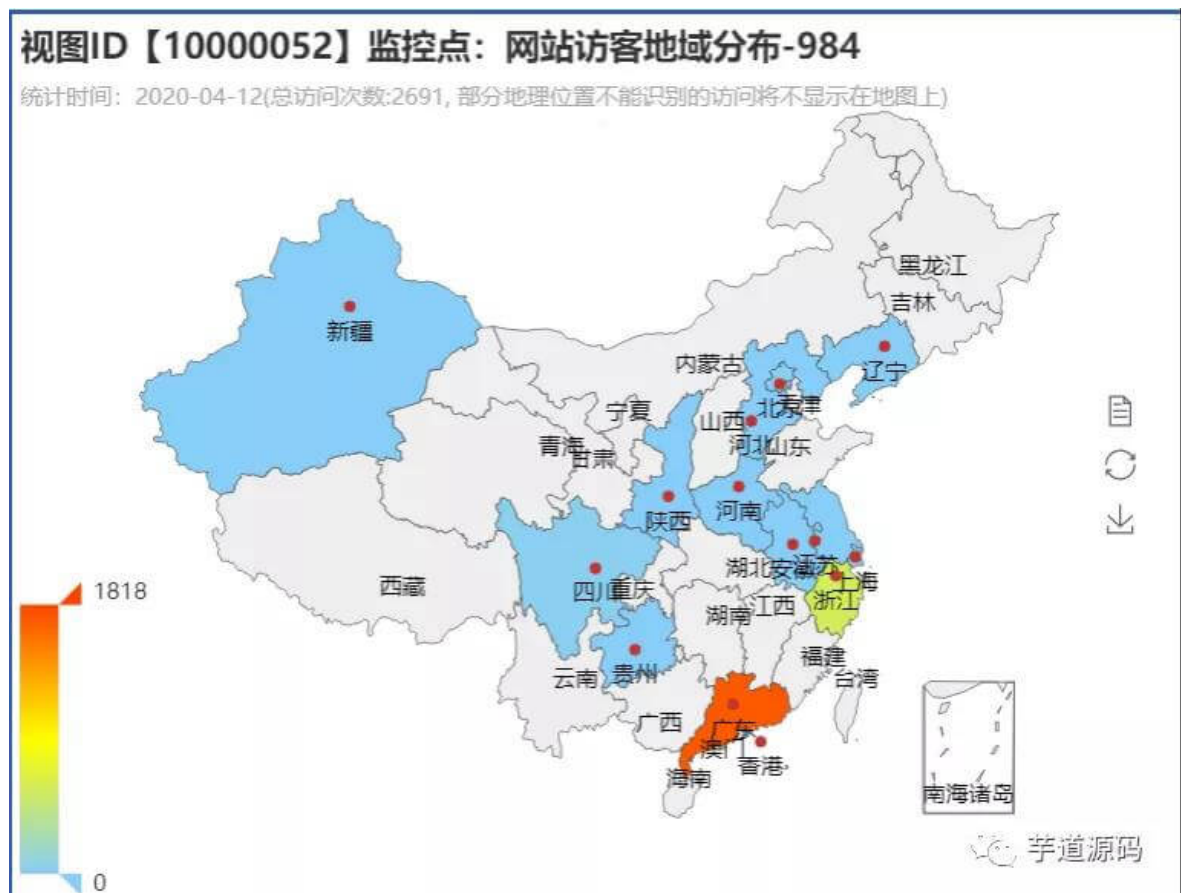
### 相比其它开源监控系统优势:

- 支持插件功能, 监控插件无需开发, 自由选择监控插件, 安装即可使用
- 集成告警功能, 支持多种告警方式
- 集成分布式日志系统功能
- 支持多种部署方式
  - a、集中部署 (全部服务部署在一台机器, 适合个人或者小团队开发者)
  - b、分布式部署 (分布式部署在多台机器, 适合小中型企业大规模监控需求)
- 支持自动化配置 (机器部署agent后自动注册到监控系统无需在控制台配置、视图根据上报自动绑定相关上报机器)
- 支持多用户访问 (子账号由管理员账号在控制台添加)
- 上报接口支持主流开发语言, 数据上报api 提供类似公共库接口的便捷

### 特色功能推荐

IP地址库: 支持通过IP地址上报时将IP地址转为物理地址, 相同物理地址归并展示一个监控API 即可轻松生成监控。

数据的物理地址分布图



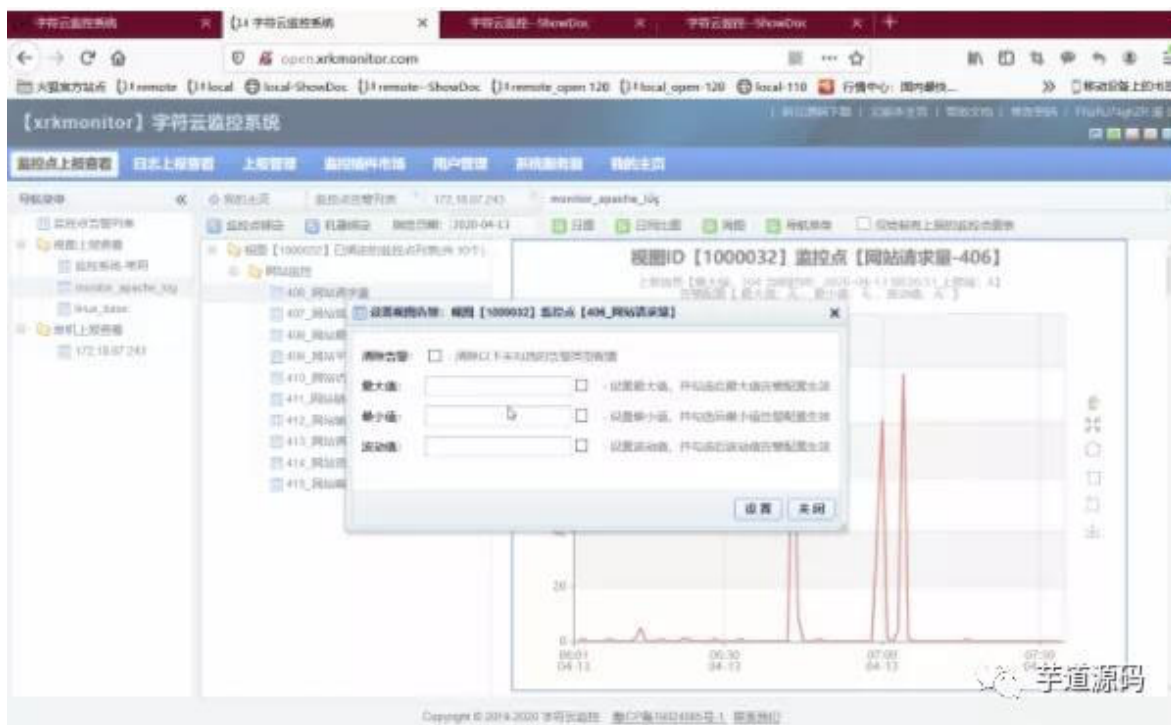
监控插件市场: 让监控成为可以复用的组件, 更多监控插件持续开发中。



分布式日志系统: 支持大规模系统日志上报, 日志上报支持频率限制、日志染色、自定义字段等高级功能, 控制台日志查看支持按关键字、排除关键字、上报时间、上报机器等方式过滤日志, 从茫茫日志中轻松找到您需要的日志。



视图机制: 监控图表支持视图定制模式, 视图可按上报服务器、监控点随意组合, 轻松定制您需要的监控视图, 并可在监控图表上直接设置告警值。



告警集成: 集成告警功能, 支持邮件、短信、微信、PC客户端等告警方式, 告警功能无需开发直接可用。



## 在线部署

安装脚本会先检查当前系统是否支持在线安装, 如不支持您可以下载源码后在系统上编译安装。

在线部署目前只支持集中部署方式, 即所有服务部署在一台机器上, 该机器上需要安装 `mysql/apache`。

安装脚本使用中文 `utf8` 编码, 安装过程请将您的终端设置为 `utf8`, 以免出现乱码。

安装脚本同时支持 `root` 账号和普通账号操作, 使用普通账号执行安装部署要求如下:

在线部署使用动态链接库, 需要在指定目录下执行安装脚本, 目录为: `/home/mtreport`  
普通账号某些目录可能无权操作, 需要授权才能正常安装  
我们强烈建议您先在本地虚拟机上执行在线安装, 熟悉安装流程后在实际部署到您的服务器上。

## 离线部署

如果在线安装失败或者需要二次开发, 可以使用源码编译方式安装。

三部完成部署:

执行 `make` 完成源码编译

进入 `tools_sh` 目录，执行 `make_all.sh` 生成部署包

在安装目录解压部署包，执行 `local_install.sh` 完成安装

使用的技术方案

`apache + mysql`(监控点数据、配置信息使用 `mysql` 存储, 支持分布式部署)

前端 `web` 控制台采用 `dwz` 开源框架

前端监控图表采用开源 `echarts` 绘制

后台 `cgi` 使用开源的 `cgi` 模板引擎 - `clearsilver`, 所有 `cgi` 支持以 `fastcgi` 方式部署

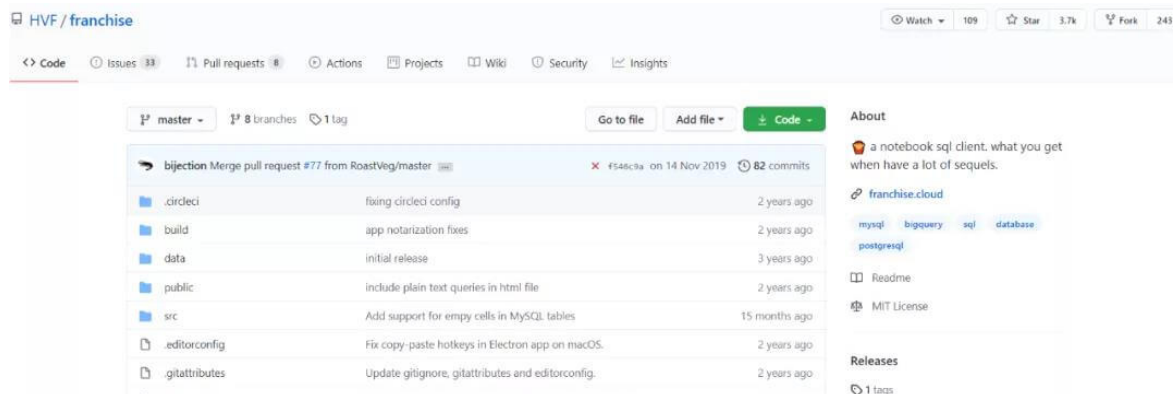
后台服务使用了开源的 `socket` 开发框架 - `C++ Sockets`

## 项目地址

官网地址: <https://gitee.com/xrkmonitorcom/open>

## SQL工具

今天，和大家推荐一款轻量级但功能强大的 SQL 工具，带有 notebook 界面。无需安装和注册，即可快速安全地使用数据——Franchise。

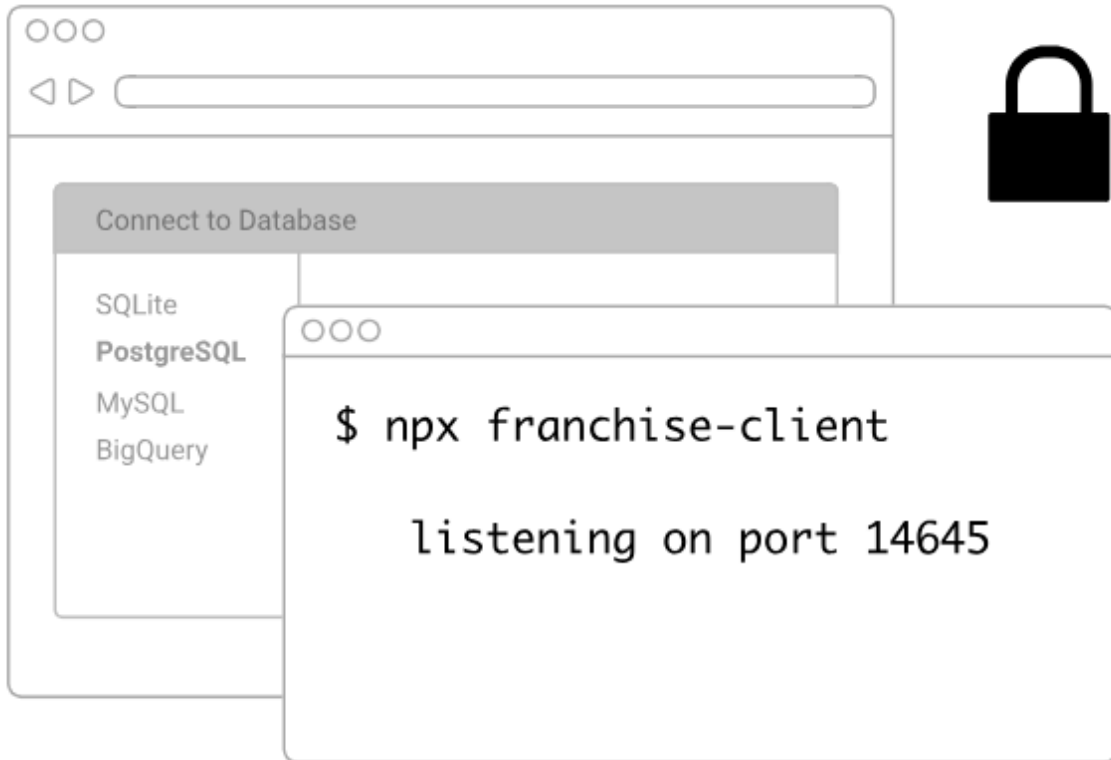


目前，Franchise在Github上标星 3.7K，累计分支 243。（Github地址：<https://github.com/HVF/franchise>）

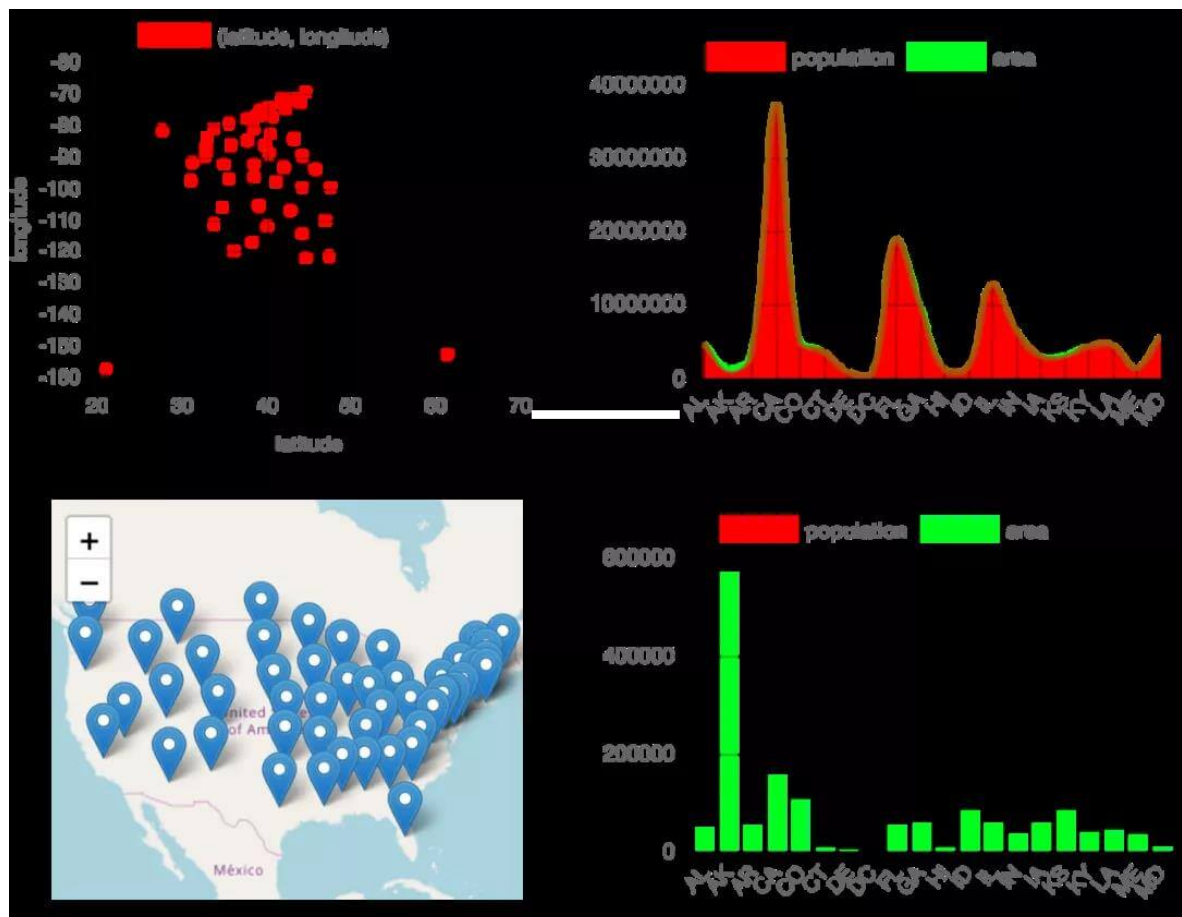
### Franchise主要功能特性特性



如果你的数据是CSV、JSON或XLSX文件，加载它就像将文件放到Franchise中一样简单。如果你想在浏览器中运行SQLite引擎的一个版本，所有的处理都在本地进行。



如果你想要连接到PostgreSQL, MySQL或BigQuery, 在你终端上运行一个命令建立数据库桥, 它将会通过你的计算机直接连接到你的数据库, 但是你的数据永远不会进入SQLite的服务器。



Franchise还为时间序列数据构建了许多可视化工具——散点图、柱状图、折线图、地图等等。

## Franchise运行方式

Franchise有在线运行版本，链接地址：<https://franchise.cloud/>，以下是在开发模式下运行Franchise的方法：

1.打开一个终端并运行

```
git clone --depth 1 https://github.com/HVF/franchise.git
```

2.cd进入项目目录

```
cd franchise
```

3.安装项目依赖项

```
yarn install
```

4.启动开发服务器

```
yarn start
```

5.打开浏览器并转到 <http://localhost:3000>

6.在franchise/src里面编辑文件，保存后，浏览器会自动重新加载

7.添加功能并发送PR。

# sqlx

## 前言

上次咱们学习了如何使用Go操作Mysql，并且实现了简单的增删改查。

但是相对来说，还有有点复杂的，可能那些大佬也都觉得繁琐叭。

就又开发出了增强版查询Mysql操作库Sqlx。

## mod文件

### go.mod

```
module sqlxDemo

go 1.14

require (
    github.com/go-sql-driver/mysql v1.4.0
    github.com/jmoiron/sqlx v1.2.0
    google.golang.org/appengine v1.6.7 // indirect
)
```

## 创建数据表

### 创建表代码

```
CREATE TABLE `userinfo` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(10) DEFAULT NULL,
  `phone` char(11) DEFAULT NULL,
  `address` varchar(64) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb4;
```

## 创建结构体

### 结构体代码

```
type Userinfo struct {
    Id      int64 `json:"id"`
    Name    string `json:"name"`
    Phone   string `json:"phone"`
    Address string `json:"address"`
}
```

## 连接数据库

### 代码



```

import (
    "fmt"
    _ "github.com/go-sql-driver/mysql"
    "github.com/jmoiron/sqlx"
)
type Userinfo struct {
    Id      int64 `json:"id"`
    Name    string `json:"name"`
    Phone   string `json:"phone"`
    Address string `json:"address"`
}
func main() {
    dsn := "root:rootroot@tcp(127.0.0.1:3306)/go_mysql_demo?charset=utf8mb4&parseTime=True"
    // 使用 MustConnect 连接的话, 验证失败不成功直接panic
    //db := sqlx.MustConnect("mysql", dsn)

    //使用 Connect 连接, 会验证是否连接成功,
    db, err := sqlx.Connect("mysql", dsn)

    if err != nil {
        fmt.Printf("connect DB failed, err:%v\n", err)
        return
    }
    db.SetMaxOpenConns(20)
    db.SetMaxIdleConns(10)
}

```

## 查询单条

我记得使用原来的方式进行查询并且绑定结构体, 是这审的。

```

//查询单条
sqlStr := "SELECT id, `name`, phone, address from userinfo where id = ?;"
var user Userinfo
err = db.QueryRow(sqlStr, 1).Scan(&user.Id, &user.Name, &user.Phone, &user.Address)
if err != nil {
    fmt.Println("查询失败", err)
    return
}

```

看第4行代码, 需要将结构体的字段一个一个点上去。

如果使用sqlx呢?

## 代码

```

//查询
sqlStr := "SELECT id, `name`, phone, address from userinfo where id = ?;"
var user Userinfo
err = db.Get(&user, sqlStr, 1)
if err != nil {
    fmt.Println("查询失败:", err)
    return
}
fmt.Println("user:", user)

```

## 执行结果

```
in: go build main.go (3) x
<4 go setup calls>
user: {1 张三 111 北京}
```

还是第4行代码，直接一个结构体扔过去，就绑定成功了。

如果表有很多字段，结构体字段也有很多，这个是很有用的。

## 查询多条

还是惯例，看看原来是怎么查的。

```
//查询多条
sqlStr := "SELECT id, `name`, phone, address from userinfo where id >= ?"
//参数同 QueryRow
rows, err := db.Query(sqlStr, 1)
//处理err
// 此处使用rows释放所有链接
defer rows.Close()
//循环整理所有数据
var userList = make([]Userinfo, 0, 10)
for rows.Next() {
    var user Userinfo
    err = rows.Scan(&user.Id, &user.Name, &user.Phone, &user.Address)
    //处理err
    userList = append(userList, user)
}
fmt.Println(userList)
```

为了方便，我去掉了err，使用伪代码处理err代替。

原来的方法，查询出来还得需要一个循环，还需要一个切片，乖乖嘞，打扰了。

来看看sqlx

## 代码

```
//查询多条
sqlStr := "SELECT id, `name`, phone, address from userinfo where id >= ?"
var userList []Userinfo
err = db.Select(&userList, sqlStr, 1)
if err != nil {
    fmt.Println("查询失败:", err)
    return
}
fmt.Println("userList:", userList)
```

## 执行结果


```
↑ ⊕ <4 go setup calls>
```

```
↓ userList: [{1 张三 111 北京} {2 李四 222 上海}]
```

```
↺
```

```
↓
```

```
Process finished with exit code 0
```

 Go语言进阶学习

还是直接扔过去，就绑定完成了，真是美滋滋。

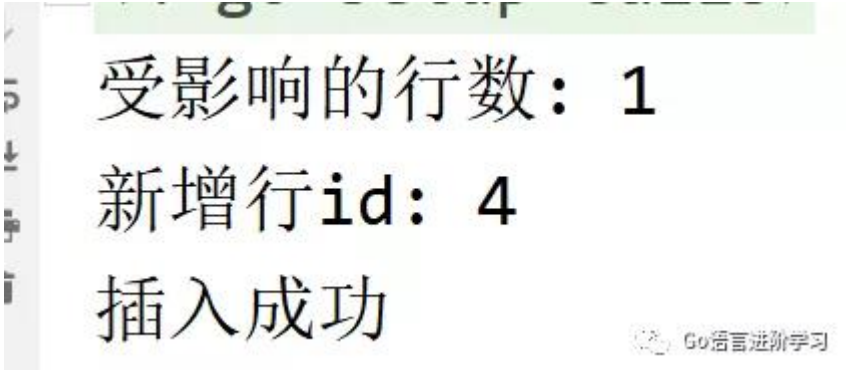
## 添加

额，添加，更新，删除，事物的话，似乎跟原来差不多，直接看代码叭。

## 代码

```
//添加
sqlStr := "INSERT into userinfo(name, phone, address) values(?, ?, ?);"
result, err := db.Exec(sqlStr, "吴彦祖", 555, "不知道哪的")
if err != nil {
    fmt.Println("插入失败", err)
    return
}
row_affect, err := result.RowsAffected()
if err != nil {
    fmt.Println("受影响行数获取失败:", err)
    return
}
fmt.Println("受影响的行数:", row_affect)
lastId, err := result.LastInsertId()
if err != nil {
    fmt.Println("新增行id获取失败:", err)
    return
}
fmt.Println("新增行id:", lastId)
fmt.Println("插入成功")
```

## 执行结果



```
受影响的行数: 1
新增行id: 4
插入成功
```

 Go语言进阶学习

## Mysql

| id | name | phone | address |
|----|------|-------|---------|
| 1  | 张三   | 111   | 北京      |
| 2  | 李四   | 222   | 上海      |
| 4  | 吴彦祖  | 555   | 不知道哪的   |

## 更新

### 代码

```
//更新数据
sqlStr := `UPDATE userinfo set name=? where id=?`
result, err := db.Exec(sqlStr, "吴彦祖666", 4)
if err != nil {
    fmt.Println("更新失败", err)
    return
}
//受影响的行数
row_affect, err := result.RowsAffected()
if err != nil {
    fmt.Println("受影响行数获取失败:", err)
    return
}
fmt.Println("受影响的行数:", row_affect)

fmt.Println("更新成功")
```

### 执行结果

```
<4 go setup calls>
受影响行数: 1
更新成功
```

## Mysql

| id | name   | phone | address |
|----|--------|-------|---------|
| 1  | 张三     | 111   | 北京      |
| 2  | 李四     | 222   | 上海      |
| 4  | 吴彦祖666 | 555   | 不知道哪的   |

## 删除

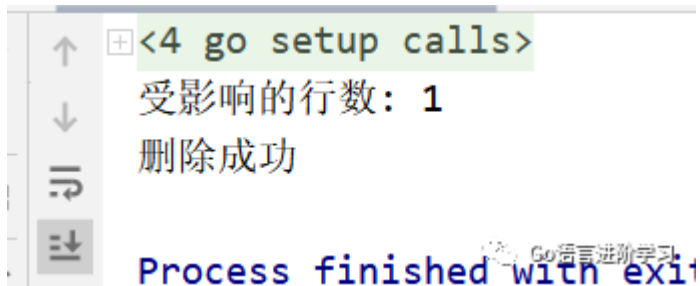
## 代码

```

sqlStr := "delete from userinfo where id = ?;"
result, err := db.Exec(sqlStr, 4)
if err != nil {
    fmt.Println("删除失败", err)
    return
}
//受影响的行数
row_affect, err := result.RowsAffected()
if err != nil {
    fmt.Println("受影响行数获取失败:", err)
    return
}
fmt.Println("受影响的行数:", row_affect)

fmt.Println("删除成功")

```



## Mysql

| id | name | phone | address |
|----|------|-------|---------|
| 1  | 张三   | 111   | 北京      |
| 2  | 李四   | 222   | 上海      |

## 事物

## 代码

```

//事物
tx, err := db.Begin()
if err != nil {
    //释回事物
    if tx != nil {
        tx.Rollback()
    }
    fmt.Println("事物开启失败")
    return
}
张三减10块Sql := `UPDATE bill set money=money - 10 where name = ?;`
result, err := tx.Exec(张三减10块Sql, "张三")
if err != nil {
    //有错误表示更是失败, 回滚原来状态
    tx.Rollback()
}

```

```

    fmt.Println(err)
    return
}
张三受影响行数, err := result.RowsAffected()
if err != nil {
    tx.Rollback() // 回滚
    return
}

李四加10块Sql := `UPDATE bill set money=money + 10 where name = ?`;
result, err = tx.Exec(李四加10块Sql, "李四")
if err != nil {
    //有错误表示更是失败, 回滚原来状态
    tx.Rollback()
    fmt.Println(err)
    return
}
李四受影响行数, err := result.RowsAffected()
if err != nil {
    tx.Rollback() // 回滚
    return
}
//都等于1表示成功, 可以提交事务, 修改数据
if 张三受影响行数==1 && 李四受影响行数==1{
    //提交事务
    fmt.Println("提交事务")
    tx.Commit()
}else{
    //有一个!=1表示没有更新成功, 可能用户不存在
    fmt.Println("失败了, 事物回滚了")
    tx.Rollback()
}
fmt.Println("事物执行成功")

```

### 执行结果

```

↑ ⊕ <4 go setup calls>
↓ 提交事务
⇌ 事物执行成功
⇓

```

### Mysql

| id | name | money |
|----|------|-------|
| 1  | 张三   | 80    |
| 2  | 李四   | 120   |

### NameExec

做增 删 改使用。

NameExec方法是通过结构体或Map绑定SQL语句，试了试，感觉用处不大，不做举例。

## NameQuery

做查询使用。

用法同上，没用，不做举例。

## 总结

其实sqlx模块，最大的改进是在查询方面，相信你也看到了，确实会比原生查询方便很多很多。

但是在其他方便，就显得捉襟见肘了，但是又说，一般还是查询场景多，查多改少。

如果你觉得文章还可以，记得点赞留言支持我们哈。感谢你的阅读，有问题请记得在下方留言噢~

## Go 表单验证器

大家好，我是欧盆索思（opensource），每天为你带来优秀的开源项目！

表单验证，Go 圈最知名的应该是 <https://github.com/go-playground/validator>，很强大，Gin 框架用的就是它。今天要介绍的，是字节跳动开源的一个类似的库，即 `go-tagexpr`。

项目地址：<https://github.com/bytedance/go-tagexpr>，Star 数 766。

官方介绍，这是一个有趣的 `go struct` 标记表达式语法，用于字段验证等。支持数据绑定和验证。

主要特性有：

- 支持多种常用运算符
- 支持访问数组，切片，字典成员
- 支持访问当前结构中的任何字段
- 支持访问嵌套字段，非导出字段等。
- 支持寄存器功能表达式
- 内置 `len`，`sprintf`，`regexp` 函数
- 支持单模和多模定义表达式
- 参数检查子包
- 使用偏移量指针直接获取值，获得更好的性能

看一个例子代码：

```
type T struct {
  A int `tagexpr:"$<0||$>=100"`
  B string `tagexpr:"len($)>1 && regexp('^\\w*$')"`
  C bool `tagexpr:"expr1:(f.g)$>0 && $; expr2:'C must be true when T.f.g>0' "`
  d []string `tagexpr:"@:len($)>0 && $[0]== 'D' ; msg:sprintf('invalid d: %v', $) "`
  e map[string]int `tagexpr:"len($)==$['len'] "`
  e2 map[string]*int `tagexpr:"len($)==$['len'] "`
  f struct {
    g int `tagexpr:"$ "`
  }
}
```



## 眼睛随鼠标移动

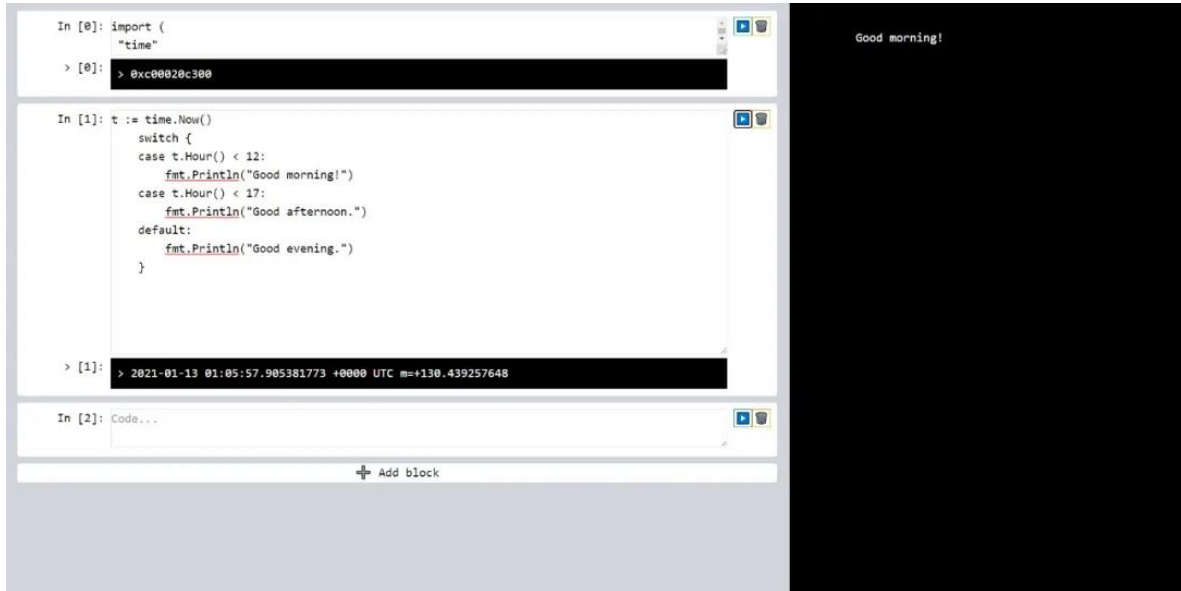
发现一个网页，是一幅 gopher 图片，图片中，gopher 的眼睛会随着鼠标的移动而移动。挺有意思的。

网址：<https://egonelbre.com/js/gopher/>，这是使用 JS，基于 canvas 实现的。实现的 JS 地址：<https://egonelbre.com/js/gopher/draw.js>。

## fiber浏览器

大家好，我是欧盆索思（opensource），每天为你带来优秀的开源项目！

如果你熟悉 jupyter-notebook，应该对这个交互式解释器感到很亲切，如下图：



这是使用 Go 语言框架 fiber 实现的，同时基于 GoLive[1] 库。

项目地址：<https://github.com/brendonmatos/gobook>，开源不到两星期。

执行：`go get -v github.com/brendonmatos/gobook` 安装，然后 `gobook` 运行，访问浏览器：<http://localhost:3000> 可以试用了。

### 参考资料

[1]GoLive: <https://github.com/brendonmatos/golive>

## 前端转Go喜欢的框架

大家好，我是欧盆索思（opensource），每天为你带来优秀的开源项目！

今天推荐一款 Go 的 Web 框架：Fiber。这是一款受 Express[1] 框架启发的 Go 框架，它基于 Fasthttp[2] 构建。旨在简化快速开发，并且注重零内存分配和性能。目前最新版本是 V2。

项目地址：<https://github.com/gofiber/fiber>，Star 数 11.2k+。

看一个简单的例子：

```
package main

import "github.com/gofiber/fiber/v2"

func main() {
    app := fiber.New()

    app.Get("/", func(c *fiber.Ctx) error {
        return c.SendString("Hello, World! ")
    })

    app.Listen(":3000")
}
```

go run 运行，在终端户会看到如下输出：

```
$ go run main.go

| Fiber v2.3.3 |
| http://127.0.0.1:3000 |
| Handlers ..... 2 Processes ..... 1 |
| Prefork ..... Disabled PID ..... 39341 |
```

打开浏览器访问：<http://localhost:3000>，看到 Hello World 表示成功！这个框架的写法感觉和 Echo 框架挺像的，属于轻量级框架。

此外，借助 fasthttp，该框架号称性能一流。

值得一提的是，这个框架很新，2020 年初开发的，但受关注度增长很快。

## Go 分布式文件系统

大家好，我是欧盆索思（opensource），每天为你带来优秀的开源项目！

今天推荐一个国产开源项目：**juicefs**，来自国内一个创业公司 **Juicedata**。  
**Juicedata** 专注于提升公有云中的大数据存储体验，为公有云存储提供标准文件系统访问接口，提升用户的数据访问和管理效率，致力于帮助工程师在云端访问和管理数据。

项目地址：<https://github.com/juicedata/juicefs>。今天刚宣布开源的。

### 看官方的介绍：

JuiceFS 是一个建立在 Redis[1] 和 S3 等对象存储之上的开源 POSIX 文件系统。它是为云原生环境设计，通过把元数据和数据分别持久化到 Redis 和对象存储中，它相当于一个无状态的中间件，帮助各种应用通过标准的文件系统接口来共享数据。

### 主要特性有：

- 完整 POSIX 兼容：已有应用可以无缝对接；
- 极致的性能：毫秒级的延迟，近乎无限的吞吐量（取决于对象存储规模）；
- 云原生：完全弹性，很容易实现存储和计算分离架构；
- 共享：可以被多个客户端同时读写；
- 文件锁：支持 BSD 锁（flock）及 POSIX 锁（fcntl）；
- 数据压缩：默认使用 LZ4[2] 压缩数据，节省存储空间。

这是今天官方宣布的文章：《2021，JuiceFS 开源啦》。另外，这里有官方中文介绍文档：

[https://github.com/juicedata/juicefs/blob/main/README\\_CN.md](https://github.com/juicedata/juicefs/blob/main/README_CN.md)。

参考资料

[1]

Redis: <https://redis.io/>

[2]

LZ4: <https://lz4.github.io/lz4>

## 图片服务器

大家好，我是欧盆索思（opensource），每天为你带来优秀的开源项目！

你是否经常有这样的需求：

- 裁剪图片
- 转换图片格式
- 旋转图片
- ...

你可能要说这些现在各种工具很容易可以实现呀。的确如此。

不过，如果这些让你通过程序实现呢？比如七牛云图片的很多功能。

今天推荐给你一个这样的项目，其实是 Go 语言的一个工具库，可以通过它快速的搭建一个图片服务器，实现类似七牛云对图片的处理功能。

项目地址：<https://github.com/pierrre/imageserver>，Star 数 1.8k+。

一个简单的示例：

```
package main

import (
    "net/http"

    "github.com/pierrre/imageserver"
    imageserver_http "github.com/pierrre/imageserver/http"
    imageserver_http_gift "github.com/pierrre/imageserver/http/gif"
    imageserver_http_image "github.com/pierrre/imageserver/http/image"
    _ "github.com/pierrre/imageserver/image/gif"
    imageserver_image_gift "github.com/pierrre/imageserver/image/gif"
    _ "github.com/pierrre/imageserver/image/jpeg"
    _ "github.com/pierrre/imageserver/image/png"
    imageserver_testdata "github.com/pierrre/imageserver/testdata"
)

func main() {
    http.Handle("/", &imageserver_http.Handler{
        Parser: imageserver_http.ListParser([]imageserver_http.Parser{
            &imageserver_http.SourceParser{},
            &imageserver_http_gift.ResizeParser{},
            &imageserver_http_image.FormatParser{},
            &imageserver_http_image.QualityParser{},
        })),
        Server: &imageserver.HandlerServer{
            Server: imageserver_testdata.Server,
            Handler: &imageserver_image.Handler{
                Processor: &imageserver_image_gift.ResizeProcessor{},
            },
        },
    })
    err := http.ListenAndServe(":8080", nil)
    if err != nil {
        panic(err)
    }
}
```

```
}  
}
```

项目提供了一个高级示例，实现了：缩放、旋转、裁剪等。

## Go直播项目

大家好，我是欧盆索思（opensource），每天为你带来优秀的开源项目！

这几年直播真的是不要太火，相关技术也很成熟。但没有接触过的人，可能不知道怎么实现的。如果你对直播技术感兴趣，今天的这个项目很适合你研究。而且是国人开发的。

项目地址：<https://github.com/gwuhaolin/livego>，Star 数：4.5k+。

这是一个简单高效的直播服务器，完全使用 Go 实现，性能高，跨平台，安装和使用非常简单。支持常用的传输协议、文件格式、编码格式。

你可以通过编译好的二进制文件：<https://github.com/gwuhaolin/livego/releases> 安装，也可以使用 Docker：

```
$ docker run -p 1935:1935 -p 7001:7001 -p 7002:7002 -p 8090:8090 -d gwuhaolin/livego
```

当然，也可以按普通的 Go 项目进行源码安装。

```
$ go get -v github.com/gwuhaolin/livego
```

### 简单使用

1. 启动服务：执行 livego 二进制文件启动 livego 服务；
2. 访问 <http://localhost:8090/control/get?room=movie> 获取一个房间的 channelkey(channelkey用于推流，movie用于播放)。
3. 推流：通过RTMP协议推送视频流到地址 `rtmp://localhost:1935/{appname}/{channelkey}` (appname默认是live)，例如：使用 `ffmpeg -re -i demo.flv -c copy -f flv rtmp://localhost:1935/{appname}/{channelkey}` 推流(下载 demo flv)；
4. 播放：支持多种播放协议，播放地址如下：
  - RTMP:`rtmp://localhost:1935/{appname}/movie`
  - FLV:`http://127.0.0.1:7001/{appname}/movie.flv`
  - HLS:`http://127.0.0.1:7002/{appname}/movie.m3u8`

使用帮助：

```
./livego -h
Usage of ./livego:
  --api_addr string    HTTP管理访问监听地址 (default ":8090")
  --config_file string 配置文件路径 (默认 "livego.yaml")
  --flv_dir string     输出的 flv 文件路径 flvDir/APP/KEY_TIME.flv (默认 "tmp")
  --gop_num int        gop 数量 (default 1)
  --hls_addr string    HLS 服务监听地址 (默认 ":7002")
  --hls_keep_after_end Maintains the HLS after the stream ends
  --httpflv_addr string HTTP-FLV server listen address (默认 ":7001")
  --level string       日志等级 (默认 "info")
  --read_timeout int   读超时时间 (默认 10)
  --rtmp_addr string   RTMP 服务监听地址 (默认 ":1935")
  --write_timeout int  写超时时间 (默认 10)
```