# 摘要

Micro是简化分布式开发的微服务生态系统。 它为开发分布式应用程序提供了基础。 Micro文档在这里为大家提供使用Micro的指南。

# 概述 overview

## 介绍 Introduction

Micro是微服务生态系统。 目的是简化分布式系统开发。

技术正在迅速发展。 云计算现在给我们几乎无限的规模，但是使用现有工具来利用这一规模仍然是困难的。 Micro尝试为开发人员首先关注这个问题。

在核心上，微Micro工具简单易用，任何人都可以轻松开始编写微服务器。 当您扩展到数百个服务时，Micro将提供管理微服务环境所需的基本工具。

### 开始

如果你想开始写微服务器，直接去查看go-micro （https://github.com/micro/go-micro）。

### 概述

提供的主要软件是Micro，一个微服务工具包。

该工具包由以下组件组成：

Go Micro - 用于在Go中编写微服务的可插入RPC框架。它提供了用于服务发现，客户端负载平衡，编码，同步和异步通信的库。

API - API网关，为微服务提供HTTP和路由请求的转换。它作为单个入口点，可以用作反向代理或将HTTP请求转换为RPC。

Sidecar - 一种语言不可知的RPC代理，具有go-micro作为HTTP端点的所有功能。虽然Go是构建微服务的伟大语言，您可能还想使用其他语言，因此Sidecar提供了将其他应用程序集成到Micro world中的一种方法。

Web - 用于Micro Web应用程序的Web仪表板和反向代理。我们认为网络应用程序应该被构建为微服务，因此被视为微服务世界中的一流公民。它的行为与API反向代理类似，但也包括对Web套接字的支持。

CLI - 与您的微服务交互的直接命令行界面。它还允许您利用Sidecar作为代理，您可能不想直接连接到服务注册表。

Bot - 位于您的微服务平台内的Hubot样式机器人，可以通过Slack，HipChat，XMPP等进行交互。它通过消息传递提供CLI的功能。可以添加其他命令来自动执行常见的操作任务。

注意：Go-micro是一个独立的库，可以独立于其他工具包使用。

### 运行

该工具包是可插拔和运行时无关的。 可以运行在笔记本电脑上，运行在docker内，可以使用kubernetes，AWS等。

IMG_256

### 更多

浏览文档以了解更多信息，查看以下资源或尝试一些示例。

（https://github.com/micro/examples）

### 资源

博客：https://micro.mu/blog/

### 赞助商

Micro的开源开发赞助商：

## 微服务 Microservices

我们来谈谈软件开发的未来。

变化正在发生，我们越来越多地走向一个以技术为动力的世界，这是每个业务的核心。 在这个时代保持竞争优势正在变得越来越困难。 组织的执行能力可能会停止，因为他们试图用低效的平台，流程和结构进行扩展。 十年老技术公司已经经历了这些痛苦，大多数人都采用相同的方法来克服这些挑战。

现在是把世界上最成功的公司的竞争优势带给其他人。 所以我们来谈谈微服务，创造竞争优势

### 什么是微服务？

Microservices是一种软件架构模式，用于将大型单片应用程序分解成较小的可管理独立服务，通过语言不可知协议进行通信，每个重点都要做一件事情。

行业专家的微服务定义：

Loosely coupled service oriented architecture with a bounded context

Adrian Cockcroft

An approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms

Martin Fowle

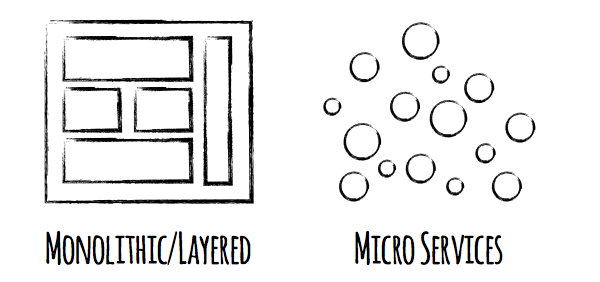
微服务的概念不是新的，这是面向服务的架构的重新定义，而是采用更全面地与unix进程和管道对齐的方法。

微服务架构的理念：

* 服务小巧精致，作为一种单一的商业目的，类似于“做一件事，做得好”的unix哲学，
* 组织文化应该包括部署和测试的自动化。 这减轻了管理和运营的负担
* 文化和设计原则应该包括失败和错误，类似于反脆弱的制度

### 为什么选择微服务？

随着组织的技术和总体规模的扩大，管理单片代码基础变得更加困难。 我们都习惯了一段时间的Twitter fail whale，因为他们试图用单片系统扩展用户群和产品功能集。 微服务使Twitter能够将应用程序分解成较小的服务，可以由许多不同的团队分开管理。 每个团队负责由许多微服务组成的业务功能，可以独立于其他团队部署。



我们已经从第一手经验中看出，微服务系统可以实现更快的开发周期，提高生产率和优异的可扩展系统。

让我们来谈谈一些好处：

更容易规模发展 - 团队围绕不同的业务需求组织并管理自己的服务。

更容易理解 - 微服务器要小得多，通常为1000 LOC或更少。

经常部署新版本的服务更容易 - 可以独立部署，扩展和管理服务。

改进的容错和隔离 - 关注分离最大限度地减少了一个服务中的问题对另一个服务的影响。

提高执行速度 - 通过独立开发，部署和管理微服务，团队更快地实现业务需求。

可重复使用的服务和快速原型 - 以微服务为根基的unix哲学使您能够重用现有的服务，并且在更快的速度上构建全新的功能

### 什么是Micro？

Micro是一个微服务生态系统，专注于提供产品，服务和解决方案，以实现现代软件驱动型企业的创新。我们计划成为有关微服务相关的虚拟资源，并将寻求使公司将自己的技术用于自己的业务。从早期阶段的原型开始到大规模的生产部署。

我们看到了业内的根本转变。摩尔定律是有效的，我们每天都在获得越来越多的计算能力。然而，我们无法充分认识到这一新的能力。现有的工具和开发实践在这个新时代不会扩大。开发人员没有被提供从单片代码基础转移到更高效的设计模式的工具。大多数公司不可避免地会以单片设计达到收益递减的程度，并且必须进行大规模的研发重组工作。 Netfix，Twitter，Gilt和Hailo都是这个的典型例子。最终建立了自己的微服务平台。

我们的愿景是提供基本的组成部分，使任何人更容易采用微服务。

## 体系结构 Architecture

Micro为微服务提供了基本的组成部分。 目的是简化分布式系统开发。 因为微服务是一种架构模式，Micro通过工具来逻辑地分离责任。

有关详细的概述，请查看架构https://micro.mu/blog/2016/04/18/micro-architecture.html上的博文。

这一部分应该更详细地说明微型建筑的构建方式以及各种libraries/repos之间的相互关系

### 工具包

API

该API充当网关或代理，以使单个入口点能够访问微服务。 它应该在基础架构的边缘运行。 它将HTTP请求转换为RPC并转发到相应的服务。

IMG_256

WEB

UI是一个web版本的go-micro，允许视觉互动进入环境。 将来，这将是一种汇总微型网络服务的方式。 它包括一种代理网络应用程序的方法。 / [name]将路由到注册表中的一项服务。 Web UI会将“go.micro.web。”（可以配置）的前缀添加到名称中，在注册表中查找它，然后将代理反向代理。

IMG_256

Sidercar

是一个HTTP接口版本的go-micro。这是将非Go应用程序整合到微型环境中的一种方式。

IMG_256

Bot

一个位于您的微服务平台内的Hubot样式机器人，可以通过Slack，HipChat，XMPP等进行交互。它通过消息传递提供CLI的功能。 可以添加其他命令来自动执行常见的操作任务。

IMG_256

CLI

Micro CLI是go-micro的命令行版本，它提供了一种观察和与正在运行的环境进行交互的方式

### Go Micro

Go-micro是微服务的独立RPC框架。 它是该工具包的核心，并由所有上述组件利用。 这里我们来看一下go-micro的每一个特征。

IMG_256

Registry

注册表提供可插拔的服务发现库来查找正在运行的服务。当前的实现是consul，etcd，内存和kubernetes。如果您的偏好不同，该界面很容易实现。

Selector

选择器通过选择提供负载平衡机制。当客户端向服务器发出请求时，它将首先查询服务的注册表。这通常返回表示服务的正在运行的节点的列表。选择器将选择要用于查询的其中一个节点。对选择器的多次调用将允许使用平衡算法。当前的方法是循环，随机散列和黑名单。

Broker

代理是pub / sub的可插拔接口。微服务是一个事件驱动的架构，发布和订阅事件应该是一流的公民。当前的实现包括nats，rabbitmq和http（用于开发）。

Transport

传输是消息的点对点传输的可插拔接口。当前的实现是http，rabbitmq和nats。通过提供这种抽象，运输可以无缝地交换。

Client

客户端提供了一种方法来进行RPC查询。它结合了Registry，Selector，Broker和Transport。它还提供重试，超时，使用上下文等。

Server

服务器是构建运行中的微服务器的接口。它提供了一种服务RPC请求的方法。

### 插件

<https://github.com/micro/go-plugins>

## 常见问题解答 FAQ

常见问题应提供最常见问题的快速答案。

1. 什么是Micro

Micro是一个专注于简化分布式系统开发的微服务生态系统

Micro是一个框架：https://github.com/micro/go-micro

Micro是一个工具包：https://github.com/micro/micro

Micro是一个社区：http://slack.micro.mu/

Micro是一个生态系统：<https://micro.mu/explore/>

Micro由开源库和工具组成，可帮助微服务开发。

go-micro- 用于编写微服务的可插拔Go RPC框架; 服务发现，客户端/服务器rpc，pub / sub

go-plugins - 包括etcd，kubernetes，nats，rabbitmq，grpc等的微型插件。

Micro - 包含传统入口点的微服务工具包; API网关，CLI，Slack Bot，Sidecar和Web UI。

github.com/micro可以找到各种其他库和服务。

1. 从哪里开始使用Micro

从go-micro开始。 readme提供了一个示例微服务器。

<https://github.com/micro/go-micro>

通过阅读入门指南或检查示例了解更多信息。

使用微型工具包通过cli，web ui，slack或api网关访问微服务器

1. 怎么使用Micro

使用go-micro编写微服务

使用micro访问他们

这有完整的实例：<https://github.com/micro/examples/tree/master/greeter>

1. 什么可以代替Consul

Etcd、Zero Dependency

1. 哪里可以使用Micro

Micro与运行时无关。 您可以随时随地运行它。 在裸机上，在AWS上，Google Cloud。 在您最喜欢的container orchestration 系统，如Mesos或Kubernetes。

实际上，对于Micro在Kubernetes的演示配置。 查看github.com/micro/kubernetes

1. API，Web和SRV服务有什么区别？

IMG_256

<https://micro.mu/docs/faq.html#web-services>

7、micro和go-kit有什么区别？

这个问题出现了很多。 micro和go-kit有什么区别？

Go-kit将自己描述为微服务的标准库。像Go一样，go-kit为您提供可用于构建应用程序的各种包。 Go-kit是您想要完全控制您如何定义服务的理想选择。

Go-micro是微服务的可插拔RPC框架。这是一个有意见的框架，其尝试简化分布式系统的通信方面，因此您可以专注于业务逻辑本身。 Go-micro是您想要快速起床和运行的好地方，同时拥有可插拔的功能，可以在没有代码更改的情况下切换基础架构。

Micro是一个微服务工具包。这就像一个微软服务的瑞士军刀，它搭载在微软上，提供传统的入口点，如http api网关，web ui，cli，slack bot等。Micro使用工具来指导您的架构中的问题的逻辑分离，推动您为公共API创建一个API层的微服务器，并为web UI单独创建一个WEB层的微服务器。

使用go-kit，您需要完全控制。使用go-micro，你想要一个自动化的框架

## 特征 Features

<https://micro.mu/docs/features.html#go-pluginshttpsgithubcommicrogo-plugins>

## 路标 Roadmap

<https://micro.mu/docs/roadmap.html>

## 资源 Resources

<https://micro.mu/docs/resources.html>

## 使用者 Users

<https://micro.mu/docs/users.html>

# 指南guides

## 安装指南

### 依赖

我们需要服务发现，可以使用组播DNS或者Consul

Consul

docker run consul

组播DNS：

将--registry = mdns添加到任何命令，例如micro --registry = mdns来列出服务

### Go-micro

被墙，所以，在github下完之后放到golang.org/x/目录。

使用Go开发的RPC微服务框架

**安装**

go get -v -u github.com/micro/go-micro

//git clone https://github.com/micro/go-micro.git

**protobuf**

如果您正在使用代码生成，您还需要使用protoc-gen-go

go get github.com/micro/protobuf/{proto,protoc-gen-go}

(Google Protocol Buffer(简称 Protobuf)是一种轻便高效的结构化数据存储格式，平台无关、语言无关、可扩展，可用于通讯协议和数据存储等领域。)

### 工具集

Micro工具包提供了访问微服务的各种方法

**安装**

go get github.com/micro/micro

**Docker**

docker pull microhq/micro

使用CLI

运行greeter服务

go get github.com/micro/examples/greeter/srv && srv

列出服务

micro list services

获取服务

micro get service go.micro.srv.greeter

请求服务

micro query go.micro.srv.greeter Say.Hello '{"name": "John"}'

## GRPC网关

本指南有助于将grpc网关与go-micro服务一起使用

<https://github.com/grpc-ecosystem/grpc-gateway>

grpc-gateway是protoc的插件。 它读取gRPC服务定义，并生成将RESTful JSON API转换为gRPC的反向代理服务器。

我们使用go-grpc写后端服务。 Go-GRPC是一个简单的包装器，围绕着go-micro和用于客户端和服务器的grpc插件。 当调用grpc.NewService它返回一个micro.Service

例子：<https://github.com/micro/examples/tree/master/grpc>

**前提**

**安装protobuf**

mkdir tmp

cd tmp

git clone https://github.com/google/protobuf

cd protobuf

./autogen.sh

./configure

make

make check

sudo make install

**安装插件**

go get -u github.com/grpc-ecosystem/grpc-gateway/protoc-gen-grpc-gateway

go get -u github.com/micro/protobuf/protoc-gen-go

## 写一个Go服务

## 写一个Go函数

## 容错

# 组件 components

# 部署 deploy

# 插件 plugins