**Go-kit**

1. **microservice go kit**

**一个分布式的开发工具集，在大型的组织（业务）中可以用来构建微服务。其解决了分布式系统中的大多数常见问题，因此，使用者可以将精力集中在业务逻辑上。**

**2. go-kit 组件介绍**

**2.1 Endpoint（端点）**

Go kit首先解决了RPC消息模式。其使用了一个抽象的 endpoint 来为每一个RPC建立模型。endpoint通过被一个server进行实现（implement），或是被一个client调用。这是很多 Go kit组件的基本构建代码块。

**2.2 Circuit breaker（回路断路器）**

Circuitbreaker（回路断路器） 模块提供了很多流行的回路断路lib的端点（endpoint）适配器。回路断路器可以避免雪崩，并且提高了针对间歇性错误的弹性。每一个client的端点都应该封装（wrapped）在回路断路器中。

**2.3 Rate limiter（限流器）**

ratelimit模块提供了到限流器代码包的端点适配器。限流器对服务端（server-client）和客户端（client-side）同等生效。使用限流器可以强制进、出请求量在阈值上限以下。

**2.4 Transport（传输层）**

transport 模块提供了将特定的序列化算法绑定到端点的辅助方法。当前，Go kit只针对JSON和HTTP提供了辅助方法。如果你的组织使用完整功能的传输层，典型的方案是使用Go在传输层提供的函数库，Go kit并不需要来做太多的事情。这些情况，可以查阅代码例子来理解如何为你的端点写一个适配器。目前，可以查看 addsvc的代码来理解Transport绑定是如何工作的。我们还提供了针对Thirft,gRPC,net/rpc,和http json的特殊例子。对JSON/RPC和Swagger的支持在计划中。

**2.5 Logging（日志）**

服务产生的日志是会被延迟消费（使用）的，或者是人或者是机器（来使用）。人可能会对调试错误、跟踪特殊的请求感兴趣。机器可能会对统计那些有趣的事件，或是对离线处理的结果进行聚合。这两种情况，日志消息的结构化和可操作性是很重要的。Go kit的 log 模块针对这些实践提供了最好的设计。

**2.6 Metrics（Instrumentation）度量/仪表盘**

直到服务经过了跟踪计数、延迟、健康状况和其他的周期性的或针对每个请求信息的仪表盘化，才能被认为是“生产环境”完备的。Go kit 的 metric 模块为你的服务提供了通用并健壮的接口集合。可以绑定到常用的后端服务，比如 expvar 、statsd、Prometheus。

**2.7 Request tracing（请求跟踪）**

随着你的基础设施的增长，能够跟踪一个请求变得越来越重要，因为它可以在多个服务中进行穿梭并回到用户。Go kit的 tracing 模块提供了为端点和传输的增强性的绑定功能，以捕捉关于请求的信息，并把它们发送到跟踪系统中。（当前支持 Zipkin，计划支持Appdash

**2.8 Service discovery and load balancing（服务发现和负载均衡）**

如果你的服务调用了其他的服务，需要知道如何找到它（另一个服务），并且应该智能的将负载在这些发现的实例上铺开（即，让被发现的实例智能的分担服务压力）。Go kit的loadbalancer模块提供了客户端端点的中间件来解决这类问题，无论你是使用的静态的主机名还是IP地址，或是 DNS的 SRV 记录，Consul，etcd 或是 Zookeeper。并且，如果你使用定制的系统，也可以非常容易的编写你自己的 Publisher，以使用 Go kit 提供的负载均衡策略。（目前，支持静态主机名、etcd、Consul、Zookeeper）

**3 目标**

在各种SOA架构中操作–预期会与各种非Go kit服务进行交互

使用RPC作为最主要的消息模式

可插拔的序列化和传输–不仅仅只有JSON和HTTP

简单便可融入现有的架构–没有任何特殊工具、技术的相关指令

**4 目标之外（不考虑做的事情）**

支持除RPC之外的消息模式（至少目前是）–比如 MPI、pub/sub，CQRS，等

除适配现有软件外，重新实现一些功能

在运维方面进行评论：部署、配置、进程管理、服务编排等

**5 依赖管理**

Go kit 是一个函数库，设计的目标是引入到二进制文件中。对于二进制软件包的作者来讲，Vendoring是目前用来确保软件可靠、可重新构建的最好的机制。因此，我们强烈的建议我们的用户使用vendoring机制来管理他们软件的依赖，包括Go kit。

为了避免兼容性和可用性的问题，Go kit没有vendor它自己的依赖，并且并不推荐使用第三方的引用代理。有一些工具可以让vendor机制更简单，包括 gb、glide、gvt、 govendor 和 vendetta。另外，Go kit使用了一系列的持续集成的机制来确保在尽快地修复那些复杂问题。