微服务初探

柯特

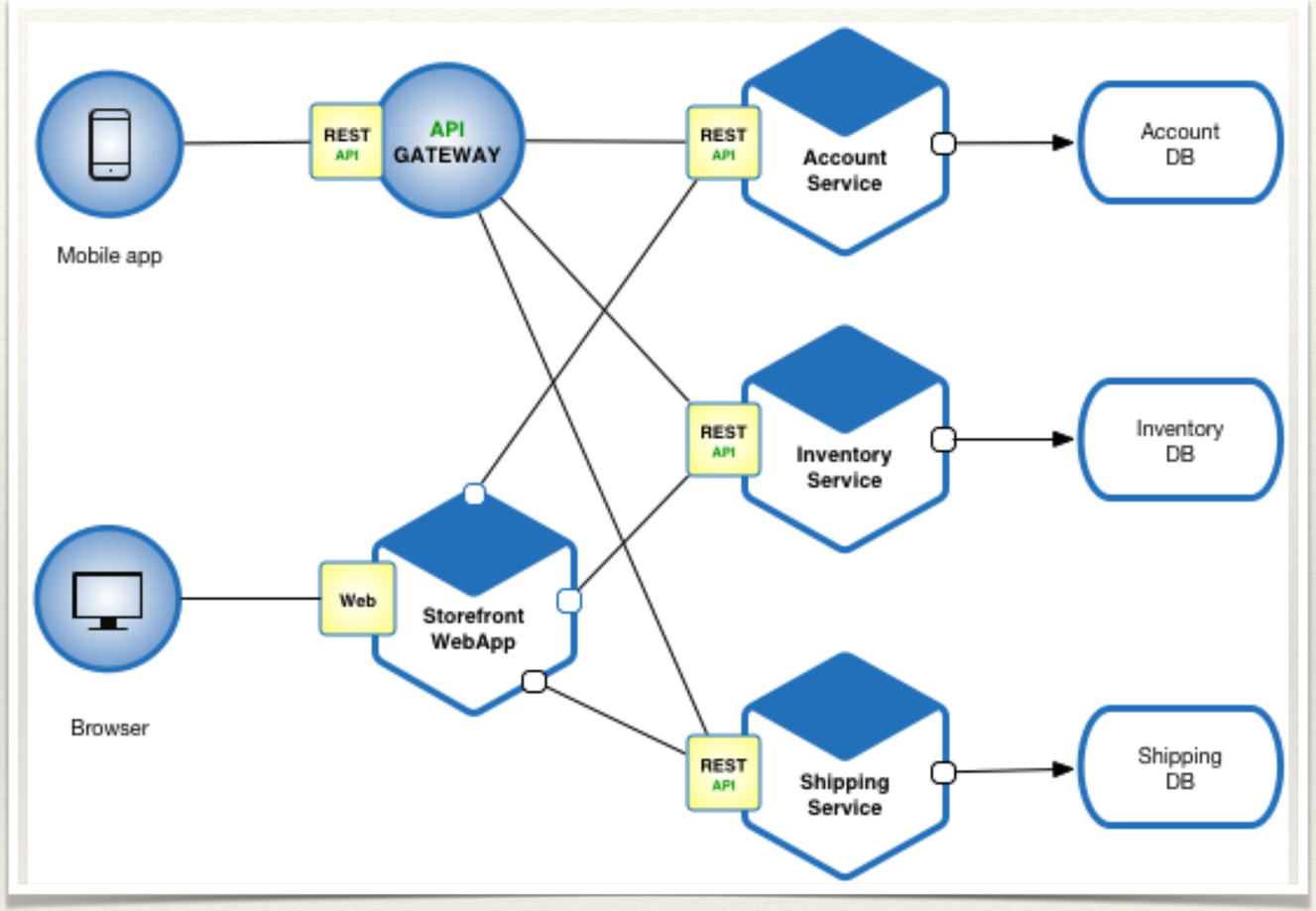
技术分享

2020-09-18

什么是微服务

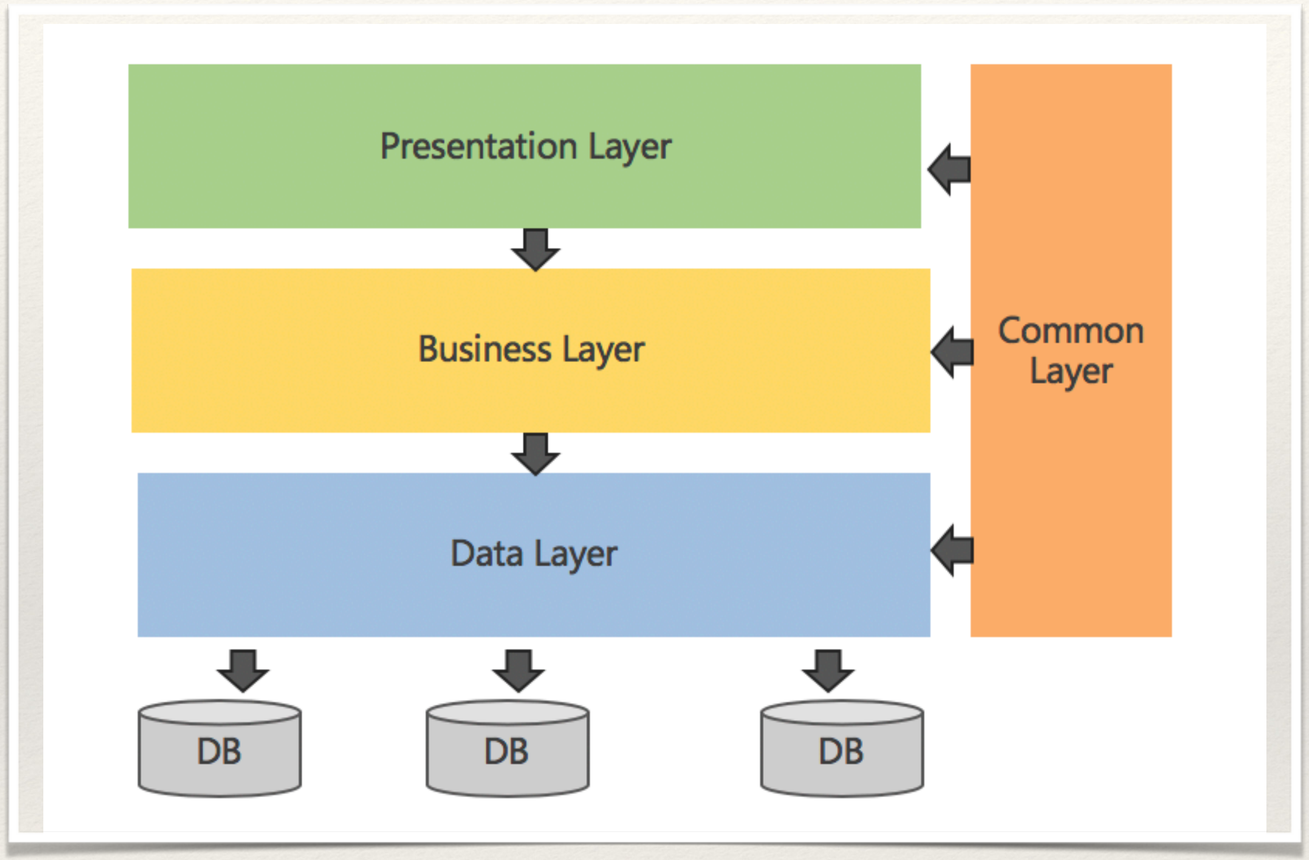
- 根据 wiki 百科 里的描述：
  微服务 (Microservices) 是一种软件架构风格，它是以专注于单一责任与功能的小型功能区块 (Small Building Blocks) 为基础，利用模块化的方式组合出复杂的大型应用程序，各功能区块使用与语言无关 (Language-Independent/Language agnostic) 的 API 集相互通信。

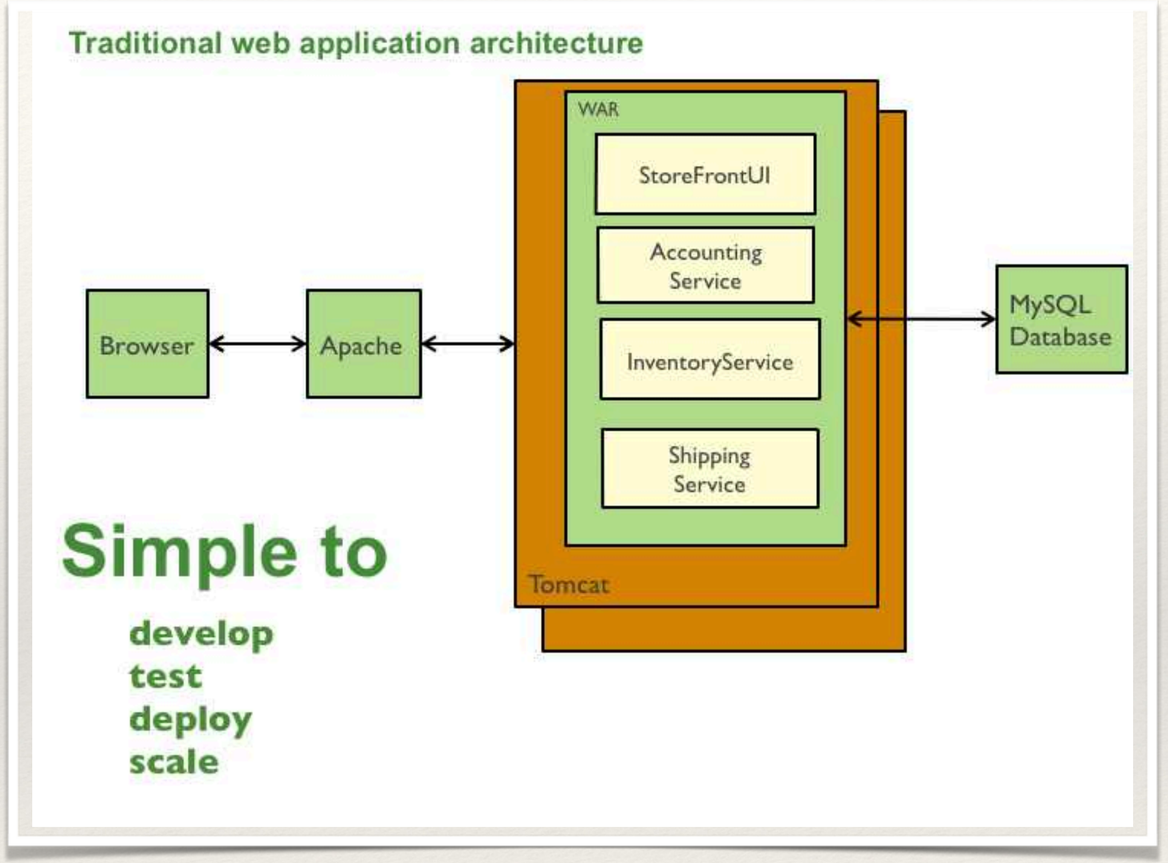
- 简单理解，就是微服务是一种软件架构体系，它将后端服务，根据业务进行拆分，使每个服务变成更小的独立单元，服务之间彼此互不影响，通过语言无关的通信机制进行沟通（比如 HTTP）。

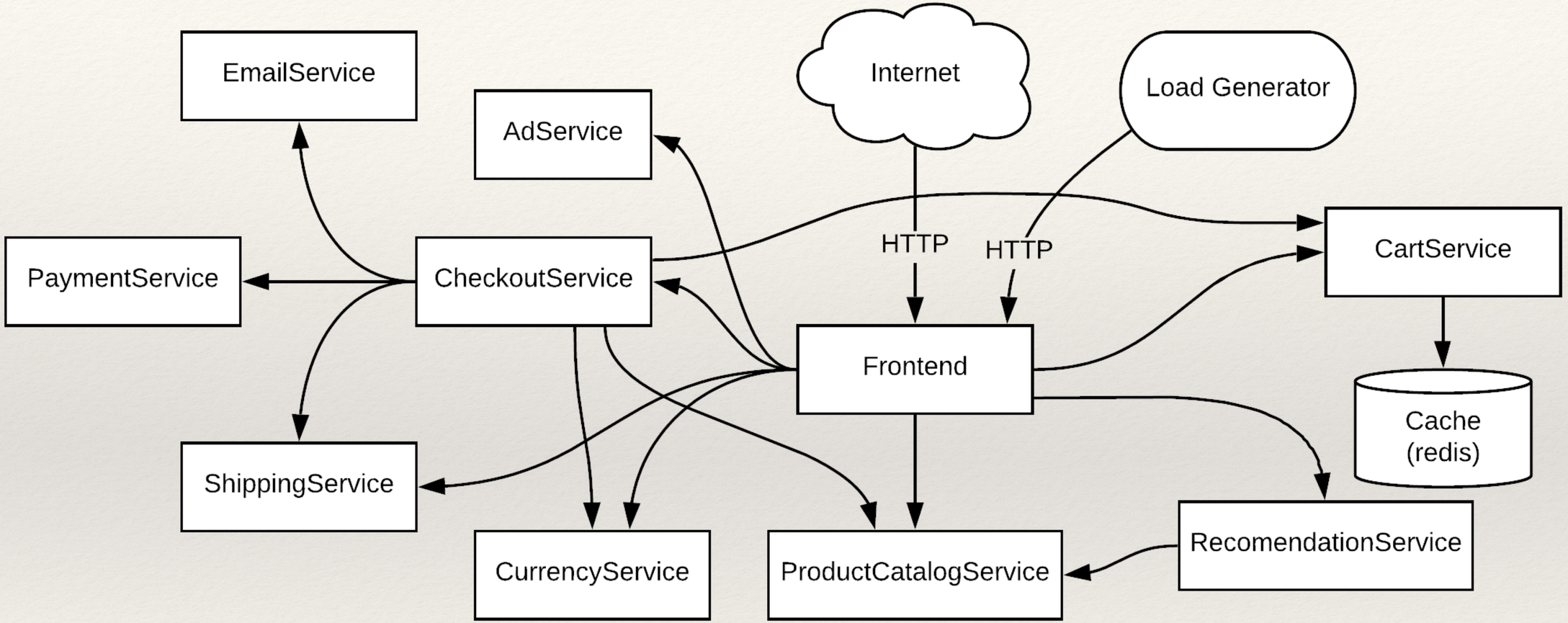


微服务的独立性

- 微服务的规划与单体式应用十分不同，微服务中每个服务都需要避免与其他服务有所牵连，且都要能够自主，并在其他服务发生错误时不受干扰。意思就是每个服务就相当于独立的应用，都可以有自己的数据库，服务资源等等。


Presentation Layer
Business Layer
Data Layer
Common Layer
DB
DB
DB


Traditional web application architecture
WAR
StoreFrontUI
Accounting Service
InventoryService
Shipping Service
Browser
Apache
MySQL Database
Tomcat
Simple to
develop
test
deploy
scale


EmailService
AdService
Internet
Load Generator
PaymentService
CheckoutService
HTTP
HTTP
CartService
Frontend
Cache
(redis)
ShippingService
CurrencyService
ProductCatalogService
RecomendationService

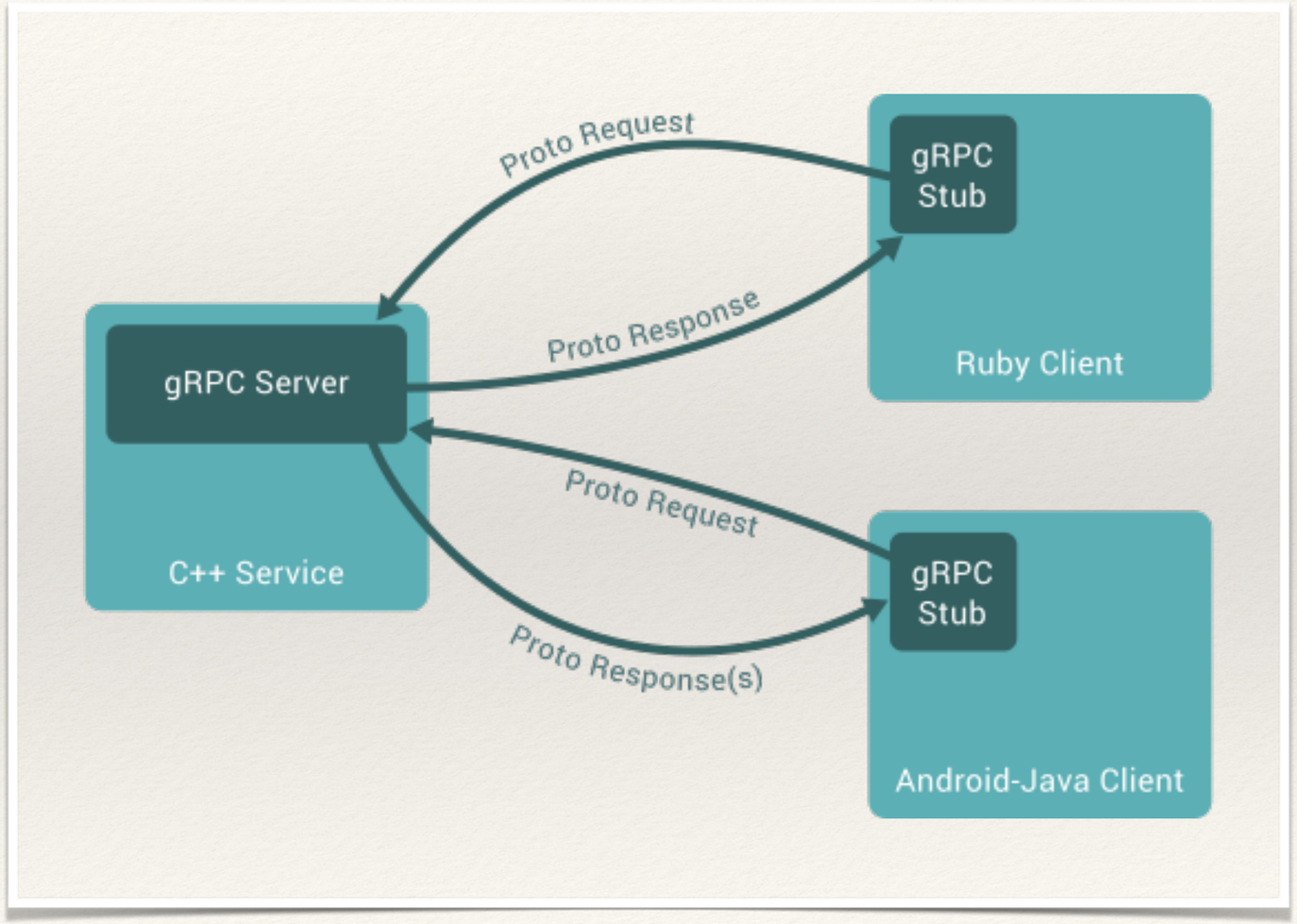
- 数据库
  从微服务的理念来看，微服务的数据应该是自己进行管理，无论数据是怎么管理，服务彼此之间的数据应该是隔离的。

- 沟通与时间广播
  微服务中最重要的就是每个服务的独立与自主，因此服务与服务之间也不应该有所沟通。倘若真有沟通，也应采用异步沟通的方式来避免紧密的相依性问题。故要达此目的，可以采用以下两种方式：
  - 沟通与事件广播：这可以让你在服务集群中广播事件，并且在每个服务中监听这些事件并作处理，这令服务之间没有紧密的相依性，而这些发生的事件都会被保存在事件存储中心里。这意味着当微服务重新上线、部署时可以重播（Replay）所有的事件。这也造就了微服务的数据库随时都可以被删除、摧毁，且不需要从其他服务中获取资料。
  - 消息队列：这令你能够在服务集群中广播消息，并传递到每个服务中。与事件存储中心近乎相似，但有所不同的是：消息队列并不会保存事件。

- 服务探索
  随着服务的不断增加，服务之间的通信可能变得复杂，这就需要进行服务管理，产生了服务中心这样的角色。单个微服务在上线的时候，会向服务探索中心（如：Consul）注册自己的 IP 位置、服务内容，如此一来就不需要向每个微服务表明自己的 IP 位置，也就不用替每个微服务单独设置。当服务需要调用另一个服务的时候，会去询问服务探索中心该服务的 IP 位置为何，得到位置后即可直接向目标服务调用。

关于通信 gRPC

- 关于微服务之间的通信，最主要的特性应该是跟语言平台无关，这个常说的就是 http 协议了。go-micro 中服务之间的通信方式主要采用 gRPC。
- 查看官网，了解各个语言对gPRC的实现



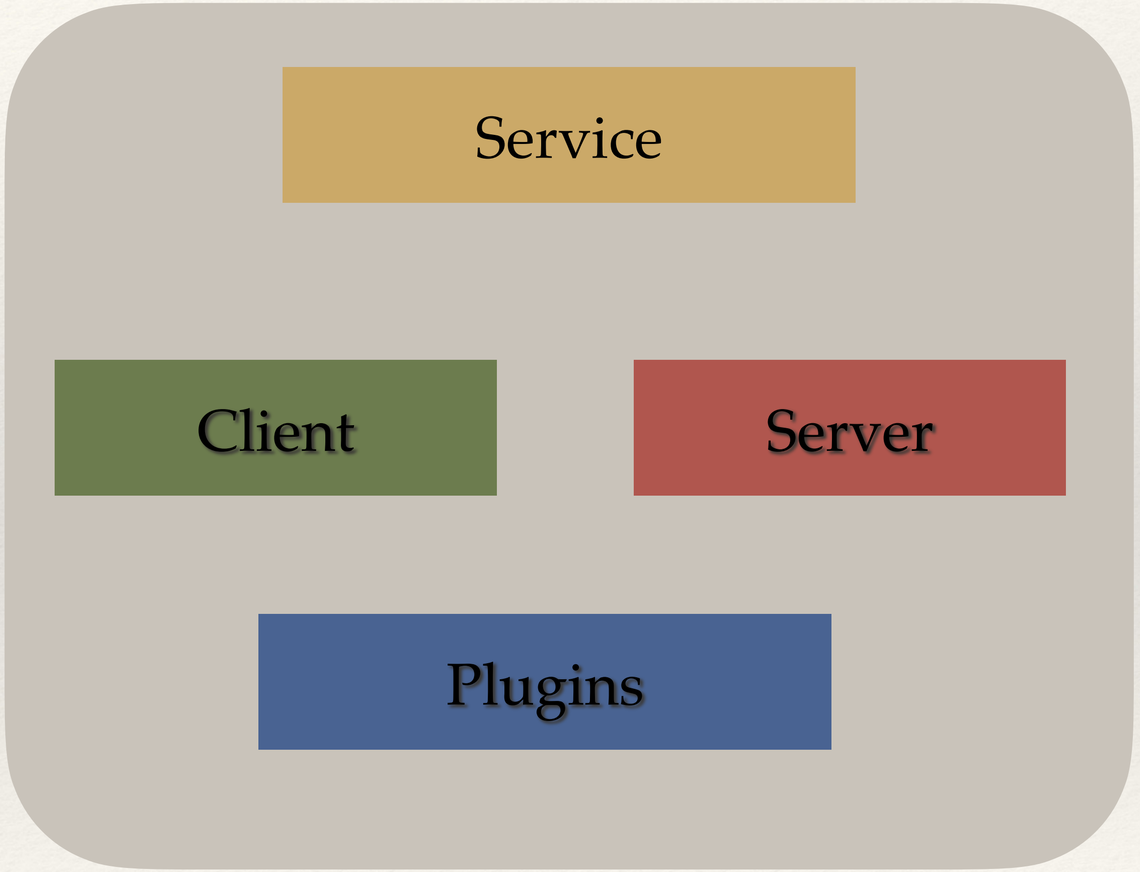
微服务的实现

- 对于微服务的这种架构模式，各个语言其实都有对应的实现：
  - Go：go-micro、go-kit
  - Node：Seneca
  - Java：Spring Cloud

go-micro

结构

- Go Micro 是基于 Go 语言实现的插件化 RPC 微服务框架。它提供了服务发现、负载均衡、同步传输、异步通信以及事件驱动等机制，并尝试去简化分布式系统间的通信，让开发者可以专注于自身业务逻辑的开发。
- 可以看到，Go Micro 框架模型的上层由 Service 和 Client-Server 模型抽象组成。Server 服务器是用于编写服务的构建块；Client 客户端提供了向服务请求的接口；底层由代理、编解码器和注册表等类型的插件组成。
- Go Micro 是组件化的框架，每一个基础功能都有对应的接口抽象，方便扩展。



特性

- **服务发现（Service Discovery）** - 通过注册组件（Registry）向注册中心注册服务信息，其它服务通过从注册中心查询服务地址及其它支撑信息（如版本号、元数据、接口等）。
- **负载均衡（Load Balancing）** - 负载均衡基于服务发现，当我们从注册中心查询出任意多个的服务实例节点时，我们要有负载均衡机制从这些节点选出一台，请求服务。
- **消息编码（Message Encoding）** - 服务之间通信需要对称编码格式，彼此才能进行编/解码。
- **Request/Response** - 同步请求，也即远程过程调用 RPC
- **异步消息（Async Messaging）** - 异步消息
- **可插拔接口（Pluggable Interfaces）** - 各大组件都支持替换插拔

Micro - 脚手架

- 为了便利程序的开发，go-micro 提供 micro 脚手架进行快速生成模板，提供了 一系列的功能。
- 比如，初始化一个服务：
  micro new --namespace=mu.micro.book --type=service —alias=test test

```
Creating service mu.micro.book.service.test in test

.
├── main.go
├── generate.go
├── plugin.go
├── handler
│   └── test.go
├── subscriber
│   └── test.go
├── proto/test
│   └── test.proto
├── Dockerfile
├── Makefile
├── README.md
├── .gitignore
└── go.mod


download protobuf for micro:

brew install protobuf
go get -u github.com/golang/protobuf/{proto,protoc-gen-go}
go get -u github.com/micro/protoc-gen-micro/v2

compile the proto file test.proto:

cd test
protoc --proto_path=.:$GOPATH/src --go_out=. --micro_out=. proto/test/test.proto
```

- Micro 脚手架生成的目录结构如图，它自动帮我们定义好了 gRPC 文件
- 最后一行命令是生成 go 语言的 gRPC 代码。

微服务与云原生

- 微服务与云原生是两种生态体系，一个侧重程序的运行环境，一注重程序的软件架构，因为它们理念契合，经常被放在一起开发，可以发挥彼此最大的优势。

DDD 领域驱动设计

- 微服务，重点在在“微”字，但是如何拆分服务，做到什么程度才叫微？DDD 的出现解决了这个问题。
- DDD 不是语言，不是框架，不是架构，而是一种思想，一种方法论，它可以分离业务复杂度和技术复杂度；DDD 也并不是一个新的事物，它是面向对象的提升，最终目标还是高内聚、低耦合。

下一代微服务 - Server Mesh

- 微服务架构的复杂性，本身引入的复杂度、学习成本，服务治理等问题。Service Mesh 出现了， Service Mesh 通过独立进程的方式隔离微服务基础组件，对这个独立进程升级、运维要比传统的微服务方式简单得多。
- Service Mesh 是用于处理服务到服务通信的专用基础架构层。云原生有着复杂的服务拓扑，它负责可靠的传递请求。实际上，Service Mesh 通常是作为一组轻量级网络代理实现，这些代理与应用程序代码部署在一起，应用程序无感知。
- Istio