FUNPLUS

Liberty -
面向云的服务架构

2022年04月01日　章北海

目录

- **背景**（为什么）
- 设计
- 实践
- 未来

背景 - 基础技术

基础技术发展：分布式，整体算力不再是瓶颈

硬件：单核 - 多核 - 虚拟机 - 容器

云服务：

AWS、GCP、阿里云、其他

国内市场2018年开始维持30%+增速

资源分配方式，收费

背景 - 业务规划

业务规划 - Service Oriented Architecture，解耦

为什么?

- 可维护：大型工程管理，**子系统复杂度可控**，问题定位，开发、迭代效率
- 灵活、定制：针对不同业务的特点，计算、io、存储、有无状态，**计算资源分配**
- 容灾：**业务隔离**
- 复用：微服务，登录、邮件、排行榜、消息频道

挑战：思维转变（封装），业务划分规划，跨业务交互，数据所有权（有状态）

背景 - 业务规划1

第一代手游的分布式服务器架构

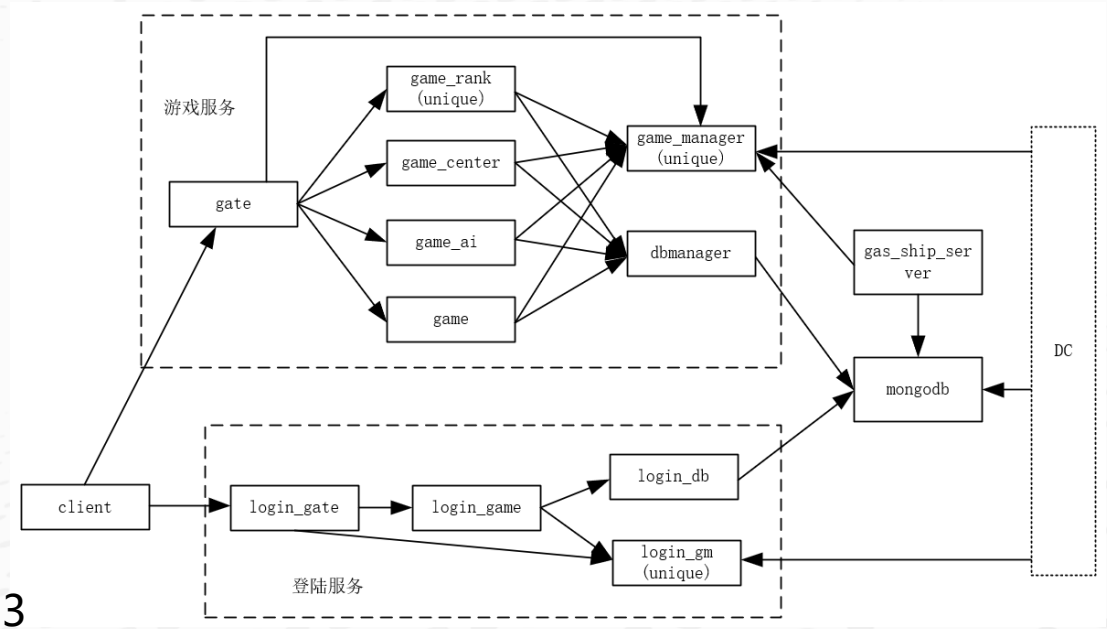
基于game的业务拆分

优点：平行扩展

缺点：

- 代码定制，代码管理、语言、lib
- 数据库，

  多库、高存储、高qps、单点、集群
- 其他中间件定制，etcd、kafka、s3
- 容灾
- 运行时更新、扩缩容



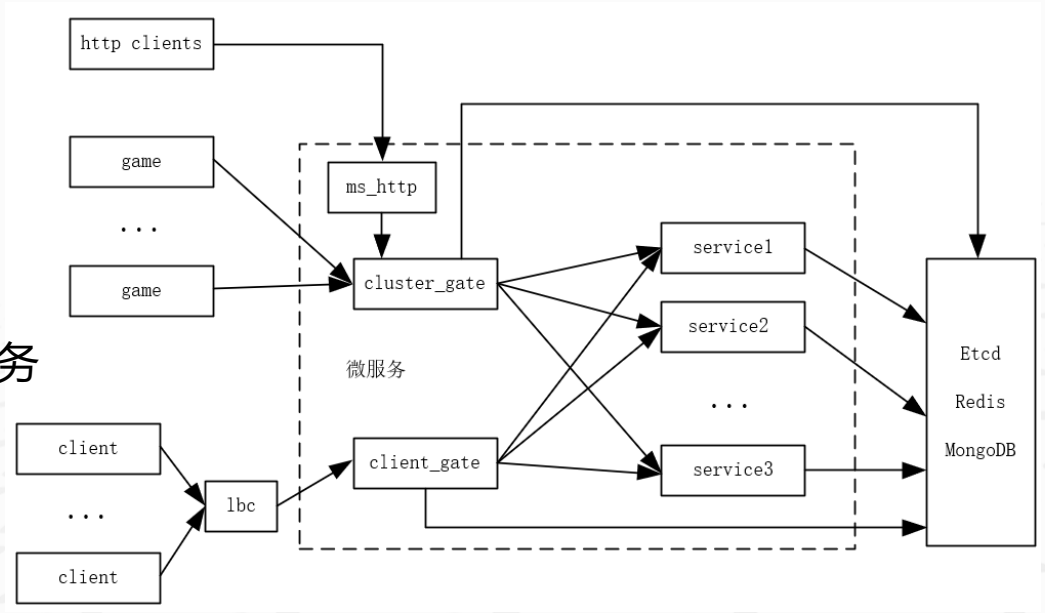
背景 - 业务规划2

微服务 - 传统服务器的强力扩展

优点：灵活，定制，容灾，扩缩容

缺点：

- 独立于主集群，架构复杂
- 无状态，应答式通信
- 1个请求1个协程，难以承载复杂业务



背景 - 初步思路

1. 基于主集群扩展，加入service节点，支持滚动更新、扩缩容，去中心

2. 基于微服务扩展，加入有状态的服务，支持服务间交互

1和2似乎走向相同的终点，1适合过渡

两者似乎都有难以摆脱的痛点

目录

设计 - 原则

简洁、性能、灵活、稳定

简洁：一切存在都是必要

性能：兼顾开发、运行效率，适用性强

灵活：自由度，可能性

稳定：更新，容灾

设计 - 特性

开发语言Golang，可扩展其他（Python3、Rust）

- 开发效率，运行效率，并发支持，多线程，完善的标准库、开发工具，社区活跃度

提供库，而不是框架

服务化，业务拆分，独立代码仓库，高度定制

容器化，云部署，灵活资源配置

无感知更新、扩缩容

网络net标准库，protobuf + msgpack

简单灵活的业务接口，method + entity

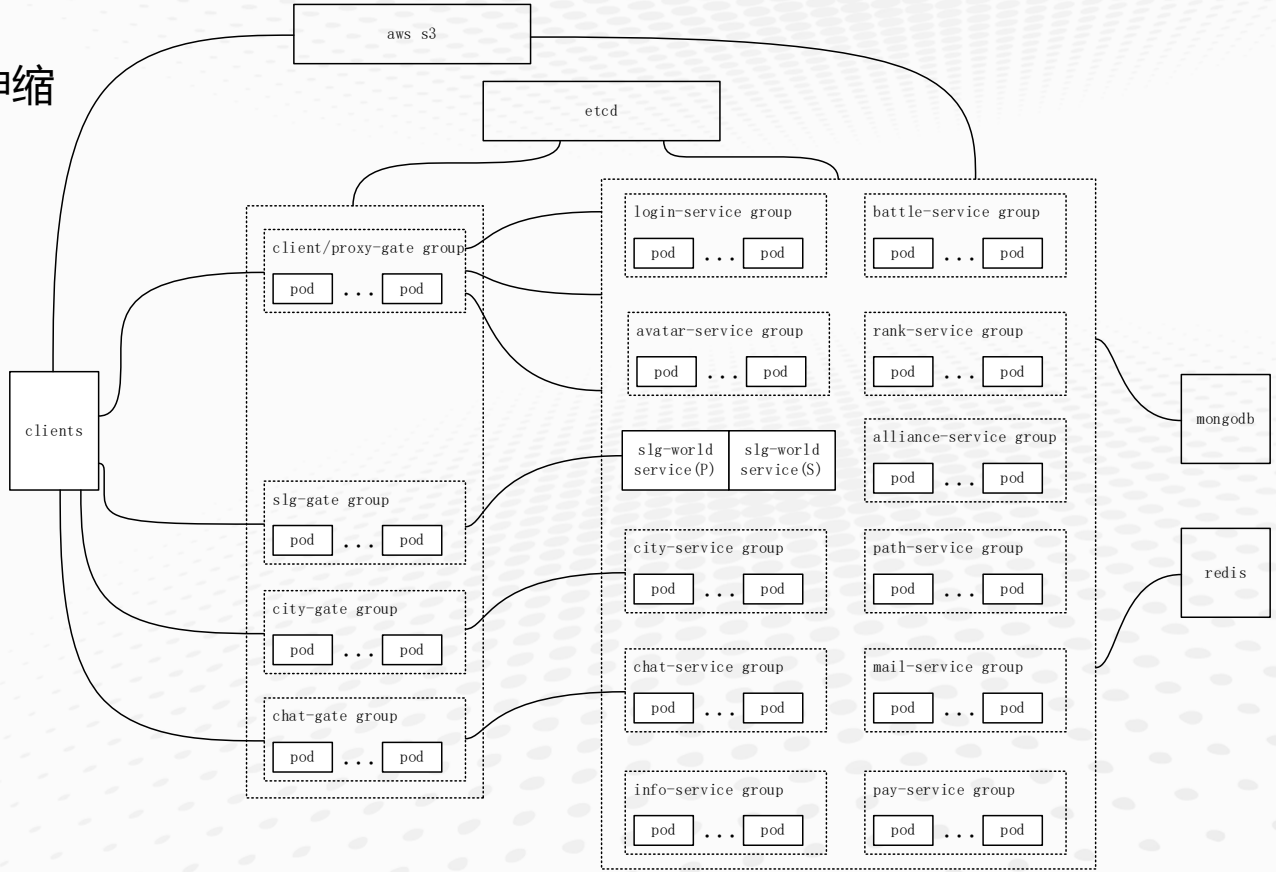
高性能，多线程计算，高效广播

设计 - 架构

业务服务化，多点、伸缩

Gate分组，下行分流

中间件定制



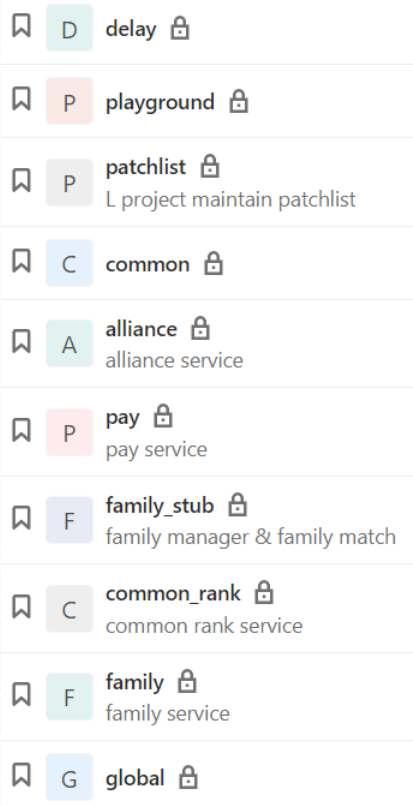
设计 - service代码仓库

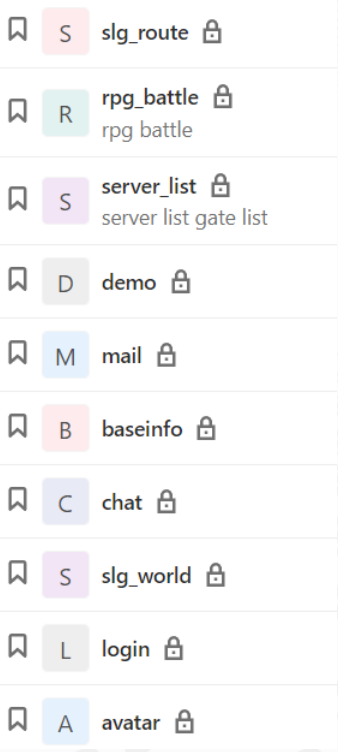
每个service一个仓库

每个service一个容器镜像

专注自身业务

框架定制：sf、http ....





设计 -引擎库

Utils：基础库

Net、Proto：网络、协议

Register：服务发现

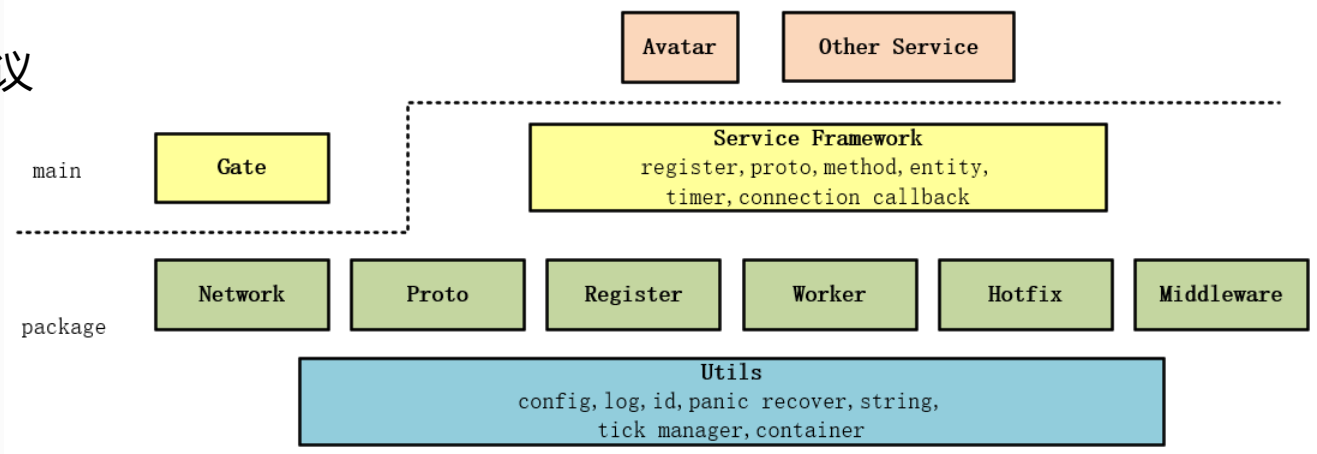
Worker：协程管理

Hotfix：热更新

Middleware：中间件

Service Framework：通用业务框架

Application：应用，gate、services



目录

- 背景
- 设计
- **特性**
- 未来

特性 - Method接口

Service对外暴露methods

应答式业务，req、rsp

独立协程、不保序

可设置消息路由，默认随机

```
reply.cb = func() {
    // callback code
}
params := map[string]interface{}{
    "logic_host": a.Data.LogicHost,
    "message": message,
    "other": other,
}
sf.CallServiceMethod("chat_service", "send_message", params,
reply, 0)
```

```
params := map[string]interface{}{
    // param fields
}
caller := sf.NewServiceMethodCaller("avatar_service",
"relogin_avatar", params, nil, 0)
caller.SetRoute(lbtproto.RouteTypeSpecific, []byte(od.Saddr))
caller.SetGaddr(gaddr)
if err = caller.Call(); err != nil {
    // handle error
}
```

特性 - Entity接口

基于mailbox的点对点通信

Entity message

单个entity内保序

主动向客户端暴露上行入口

```
a.stub := sf.NewServerEntityStub(addr, id)

params := &com_rpc.SlgAttrParam{
    // param fields
}
a.stub.CallServerMethod("sync_slg_attr", params)
```

```
a.stub := sf.NewServerEntityStub(addr, id)

params := &com_rpc.SlgAttrParam{
    // param fields
}
a.stub.CallServerMethod("sync_slg_attr", params)
```

特性 - Entity设计

线程安全规则

特性 - 协程管理

Worker设计，多线程

特性 - 高效广播

Filter msg，Group msg

特性 - 更新、扩缩容

线程安全规则

目录

- 背景
- 设计
- 实践
- **未来**

实践 - 线程安全问题

线程安全，数据隔离，加锁

实践 - panic问题

Recover策略

常见panic

实践 - 多线程计算

复杂业务举例，slg-world

实践 - lua协议decode

Msgpack decode lua