后端架构设计思想

需求分析

1、 承载量需可伸缩，可应对高并发，扩容方便

2、 高可用，数据安全

3、 维护方便

棋牌游戏从技术原理角度看类似 IM 即时通信类业务，因此完全适用当下流行的微服务。

微服务通俗的说法

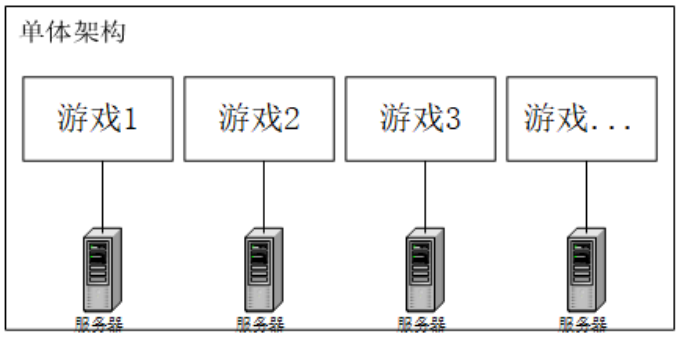
原先的单体应用讲的面向对象、模块化，模块需在同一台服务器甚至运行在同一进程。而微服务则将每一个模块单独拿出来运行，不需要依赖别的模块，实现了松耦合。

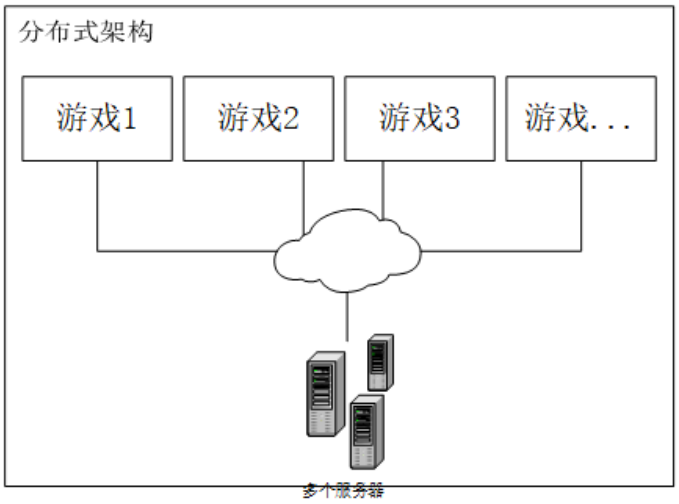
例如推导图，其中模块、服务可以全部装在一台服务器，也可以装在不同服务器，也可以多台服务器装同一个模块。这样的架构可以轻松实现分布式，解决性能瓶颈。例如斗地主服务访问量大，可以单独设一台或者多台服务器来专门支撑斗地主服务；用户充值提现等并发小访问少，可以多个服务放同一台服务器。

注册中心 Eureka 可实时监控所有服务的性能指标，扩容部署也非常方便，这样的特性尤其适合做包网。

单体架构和微服务区别

微服务化之后，所有服务之间都是松耦合，代码更加清晰。例如要改用户充值服务的逻辑，刚请来的程序员只需要研究用户充值模块就能修改，不需要把整个后端代码都看一遍。多个程序员在同时开发不同模块的时候也不用担心受别人的影响，只要做好自己的就行。





包网流程

项目	单体架构	微服务分布式架构
前端换皮	✓	✓
苹果包签名	✓	✓
购买域名	✓	✓
环境部署	购买服务器、配置环境	免部署，或扩容部署
部署项目	单独部署	后台设置
差异化（代理模式、人民币游戏币、具体哪些游戏、接入什么支付）	改程序，单独部署	后台设置
攻击防护	单独部署	免部署
维护	每台服务器登进去看	健康检测系统
故障	发现->技术员单独处理	服务短路机制+多节点高可用

包网效果

项目	单体架构	微服务分布式架构
玩家数据	独立	有总数据和代理游戏内的数据
游戏活跃	机器人	机器人+可跨游戏互通
赢率算法	单个游戏内	为更多维度计算提供数据支持
运营数据	一个个后台打开看	总报表多维度大数据计算

技术难度

推导图上所列的均为当下最流行的技术栈，非常成熟。对于编程技术要求不高，难在理解架构思想。但代码量会比单体应用大一些。

语言选择

1、 要求执行效率高，所以只能用编译型语言，python，nodejs 这类脚本语言 pass 掉

2、 要求开发效率高，所以 C\C++也 pass 掉

现在只剩下 Java 和 Golang。

这两个语言执行效率差不多，Go 稍微高一些，java 发展时间长，语言特性严谨，稳定性更高。开发效率毋庸置疑 Go 比 Java 高，作为新出的语言，编程思想更符合当下的潮流。

所以，如果做单体应用，用 Golang 是比较合适的，毕竟更简单。但如果做分布式则只能用 Java，原因有下：

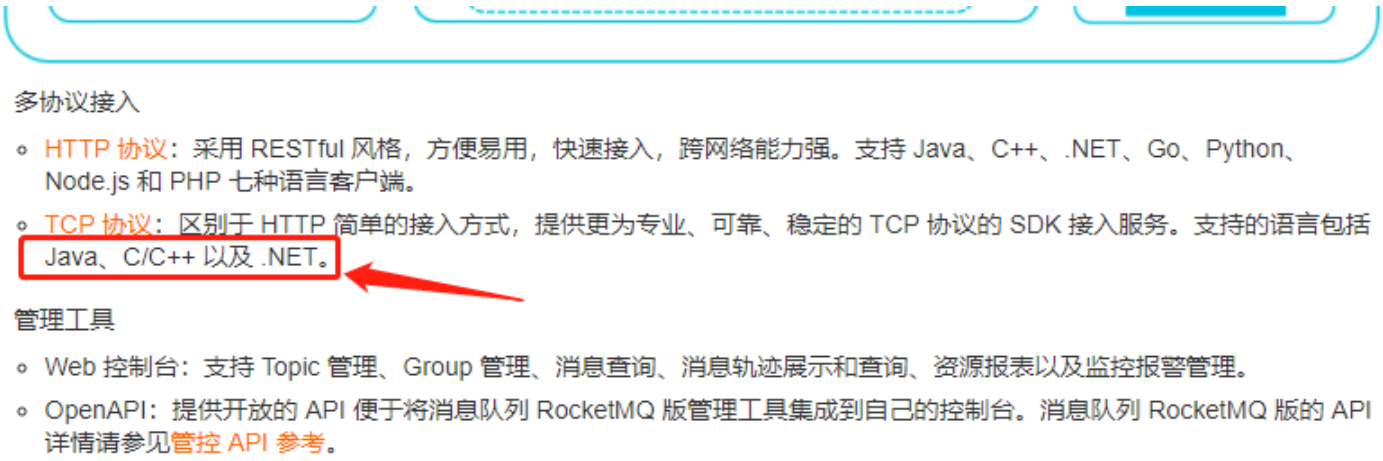
1、 Go 的发展时间短，生态和 Java 相比完全不是一个量级的，Go 的 RPC 框架寥寥无几，仅有的几个中文文档都找不到。而 Java 的微服务框架选择多而且非常成熟，例如推导图中的技术栈全部基于 Java Spring Cloud。

2、 Java 在国内的应用广泛，当前主流技术栈大部分是基于 Java 的，所以基础设施也建设得非常好。例如推导图中的日志管理、消息队列，甚至整个微服务框架都可以不用自己搭，阿里云有现成的

服务


阿里云
中国站
全部
域名
商标
云数据库 RDS MySQ
云数据库 MySQL 版 | 性能测
最新活动
产品分类
企业应用中心
解决方案
云市场
消息队列 RocketMQ 版
消息队列 RocketMQ 版是阿里云基于 Apache RocketMQ 构建的低
件。该产品最初由阿里巴巴自研并捐赠给 Apache 基金会，服务于
一交易核心链路的官方指定产品，支撑千万级并发、万亿级数据洪
点击免费开通
新用户9.9元包月起
帮助文档 >>
消息队列 MQ / 消息队列 RocketMQ...
产品优势
消息队列 RocketMQ 版
消息队列 Kafka 版
消息队列 AMQP 版
微消息队列 MQTT 版
据与消息，从未如此简单
级，解决分布式事务最

这些现成的服务大多都是基于 Java 技术栈搭建的，目前很多还不支持 Golang



所以就目前而言，如果要做微服务必须用 Java 做架构，具体服务（如具体游戏的服务端）用 Go、Java、

C++都行。

数据存储

用非关系型数据库处理日志、玩家行为记录，其它数据用关系型数据库。架构中大部分节点放国内，但数据库放国外，至少敏感数据（能统计到运营数据的数据）一定放国外，通过分布式 Redis 解决国内访问性能问题，国内节点通过自定义加密 tcp 协议访问国外数据。

关于胜率算法

微服务化之后，有一个服务专门计算胜率（暂称算法服务），玩家每开一局游戏，游戏服务都会向算法服务请求一个该玩家的胜率范围，这个值可以是固定的，可以是以前的算法，也可以尝试通过大数据寻求更精准合理的值，毕竟现在的架构已经将用户行为数据作为单独的模块，可供大数据算法提供数据支持。

修改算法不需要改动游戏，毕竟它是一个独立的服务，为完善算法提供了更好的环境支持。