Spring Cloud & SaaS

实战经验分享

A.I.Roland

目录

CONTENTS

SaaS漫谈

架构设计

实战经验分享

总结

SaaS漫谈

- SaaS模式是什么？
- SaaS模式有哪些特殊性？

SaaS模式是什么？

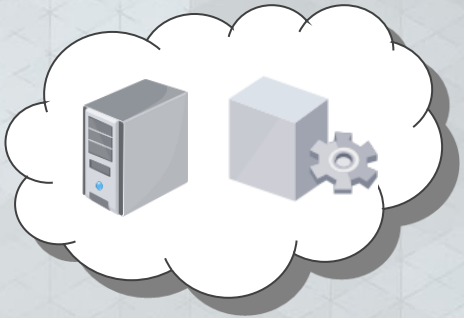
传统软件模式

软件产品

去客户现场实施

通常部署
在局域网

SaaS模式



产品在云服务器上

按需付费

租户

SaaS多租户数据库方案

完全隔离

独立数据库

隔离+共享

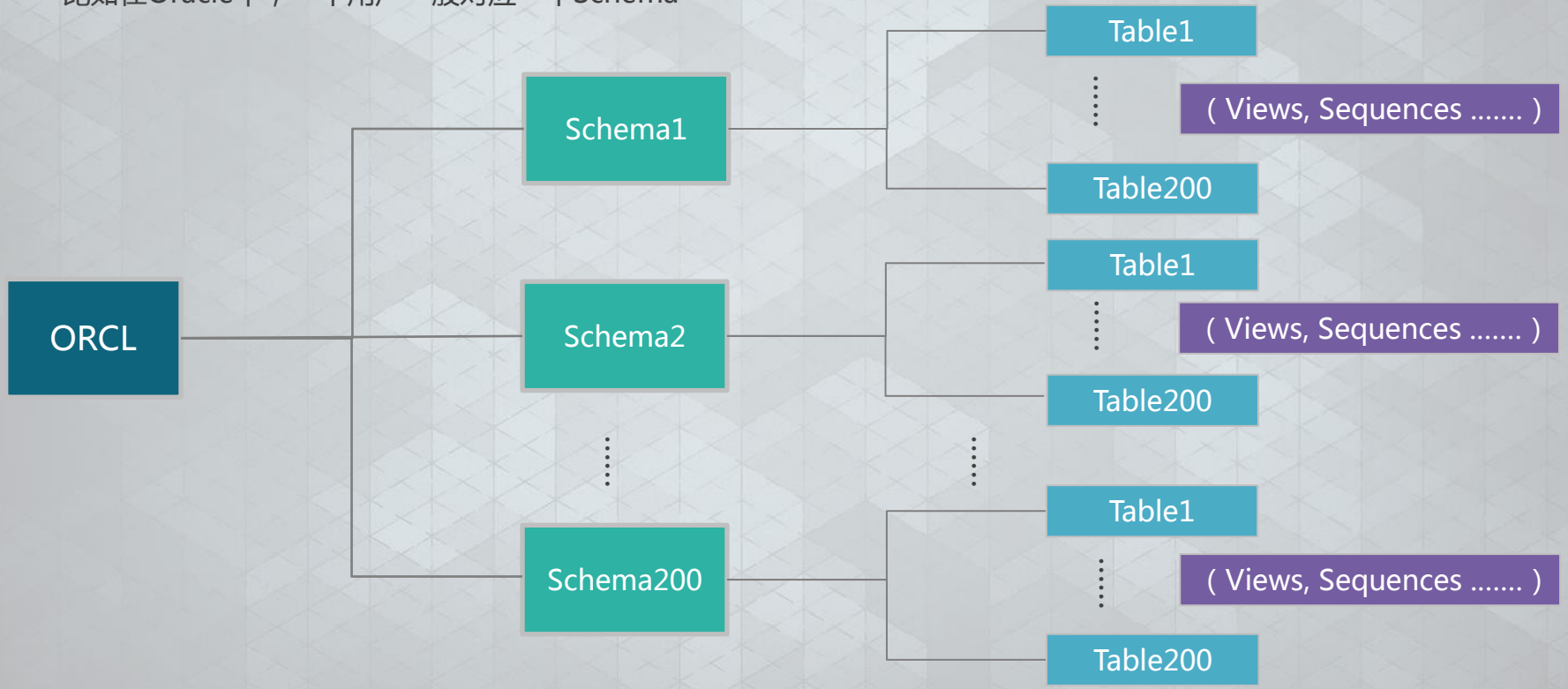
共享数据库
独立Schema

完全共享

共享数据库
共享数据表

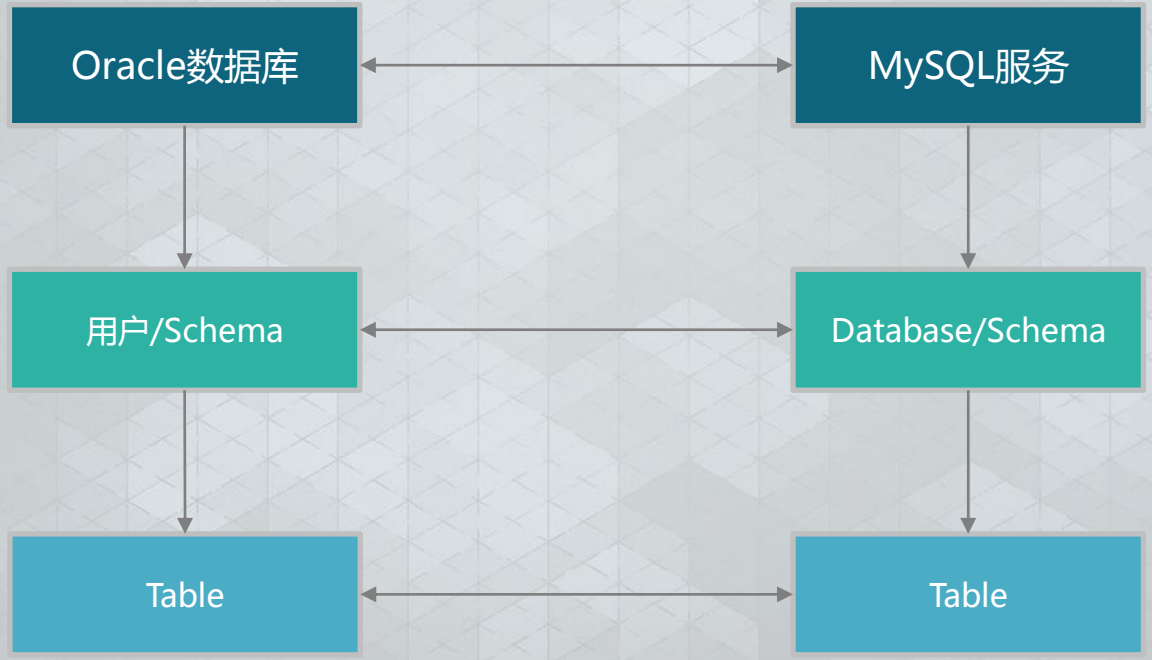
什么是Schema？

- 数据库中的Schema，为数据库对象的集合
- 比如在Oracle中，一个用户一般对应一个Schema



MySQL中的Schema

- 对MySQL来说，Schema并不是Database的下级，而是等同于Database。比如执行create schema test，和create database test是一样的；
- Oracle与MySQL的数据库层级对应如下：



独立Schema模式的优点和问题

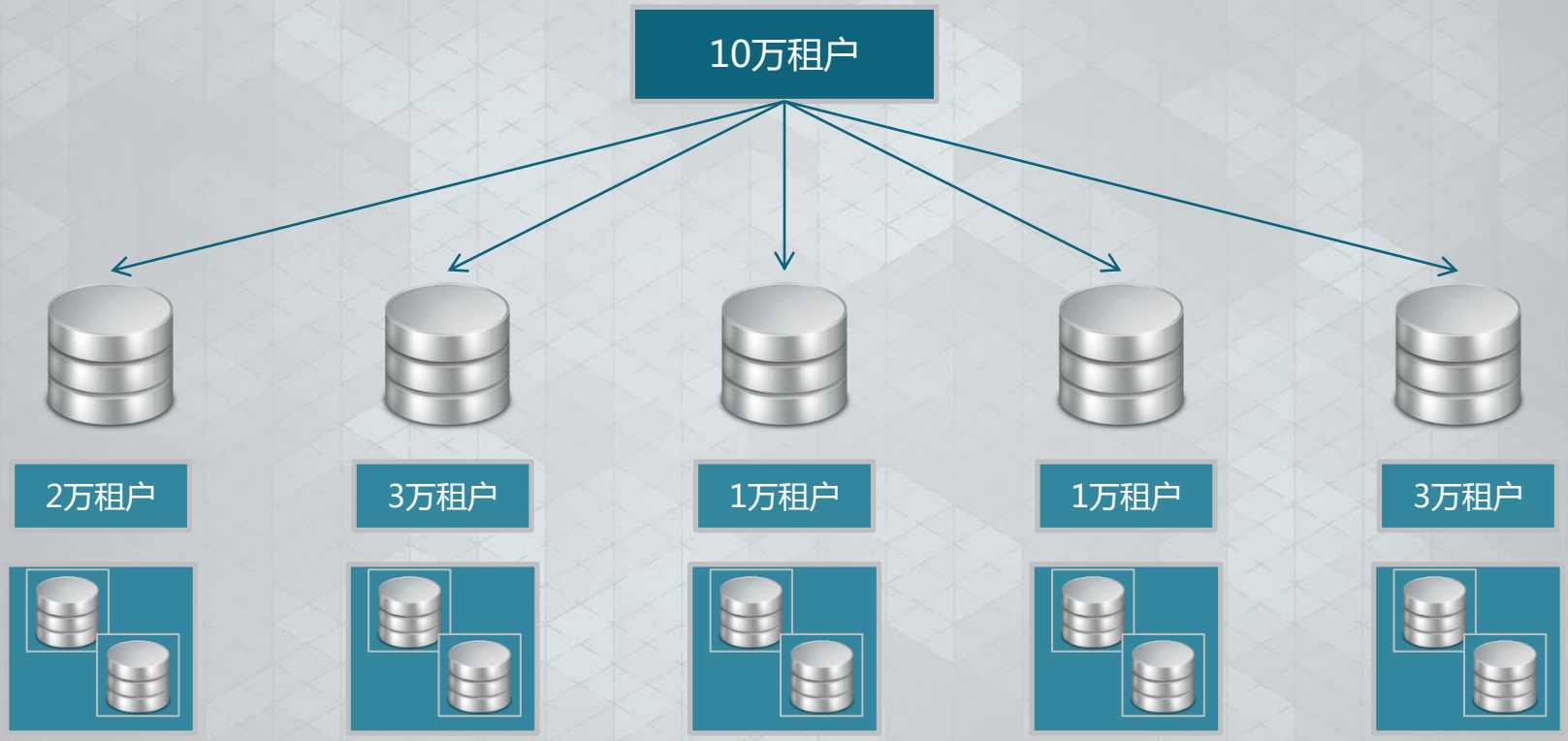
1 独立Schema模式的优点

- **高独立性**：每个租户都拥有自己的库，与其他租户是隔离的；
- **高可扩展性**：可以方便的进行横向扩展和数据迁移；
- **业务开发简单**：开发时只需要考虑单租户的业务逻辑即可，通过切换Schema来达到多租户的效果，联查的表更少；
- **定制化服务**：用户可以定制个性化服务，不影响其他租户；

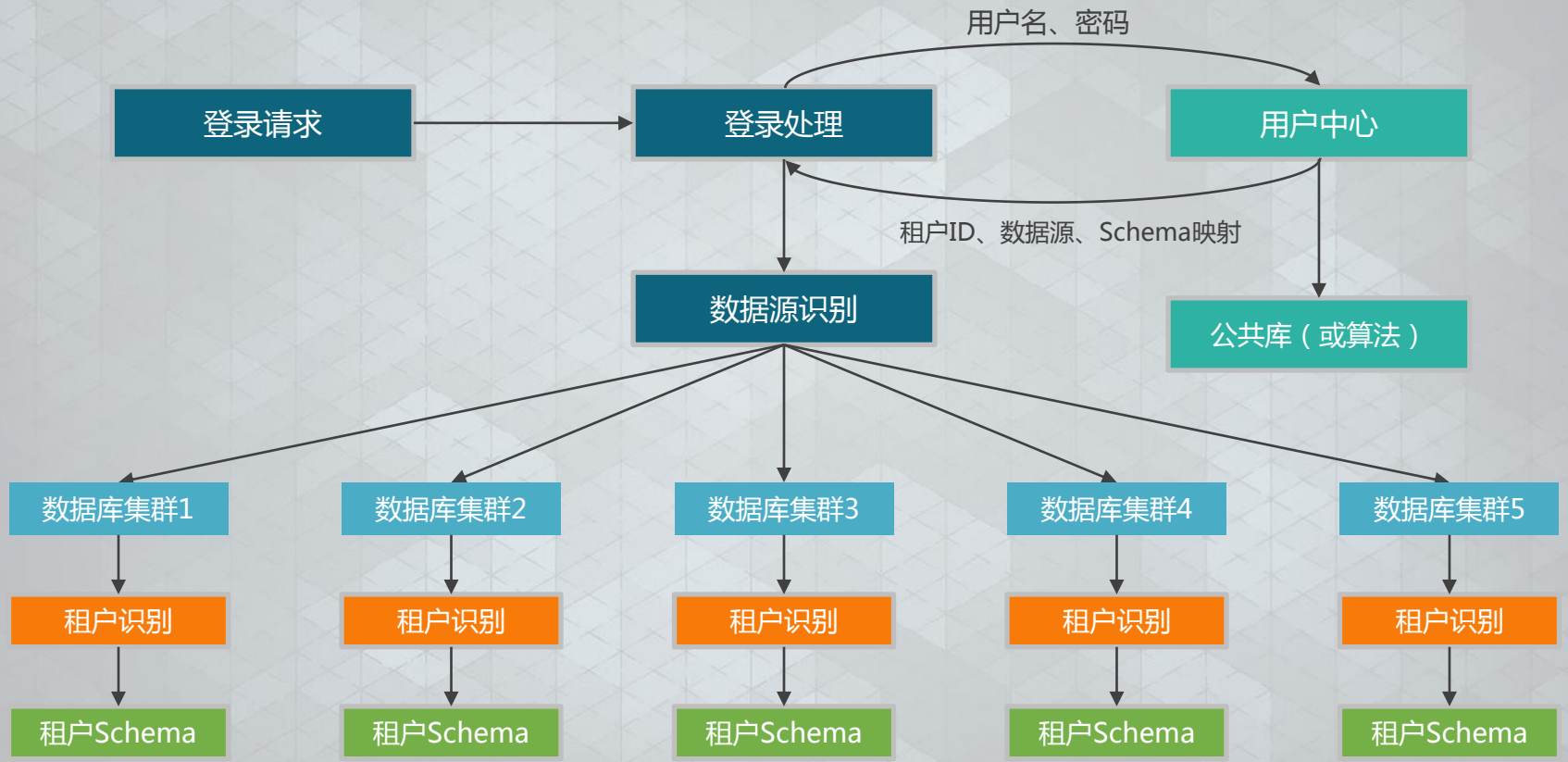
2 独立Schema模式存在的问题

- 数据库越来越多怎么办？假设有10万个租户，就有10万个库，单个服务器肯定无法承受。
- 如此多的数据库，如何进行表的更新与维护？
- 租户的数据都隔离开了，进行整体数据分析的时候怎么办？

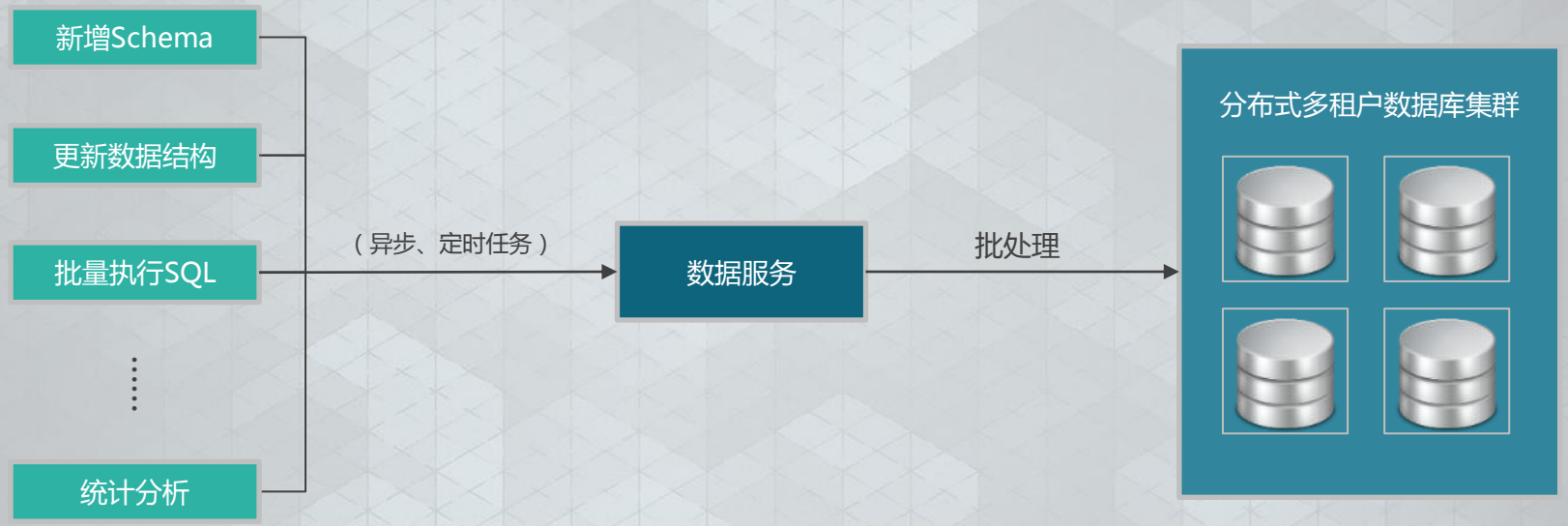
分布式多租户数据库集群



如何定位租户Schema？



独立的数据服务



架构设计

- 微服务的拆分原则
- 分层设计
- 业务架构设计
- 拓扑举例

微服务的拆分原则

分层设计

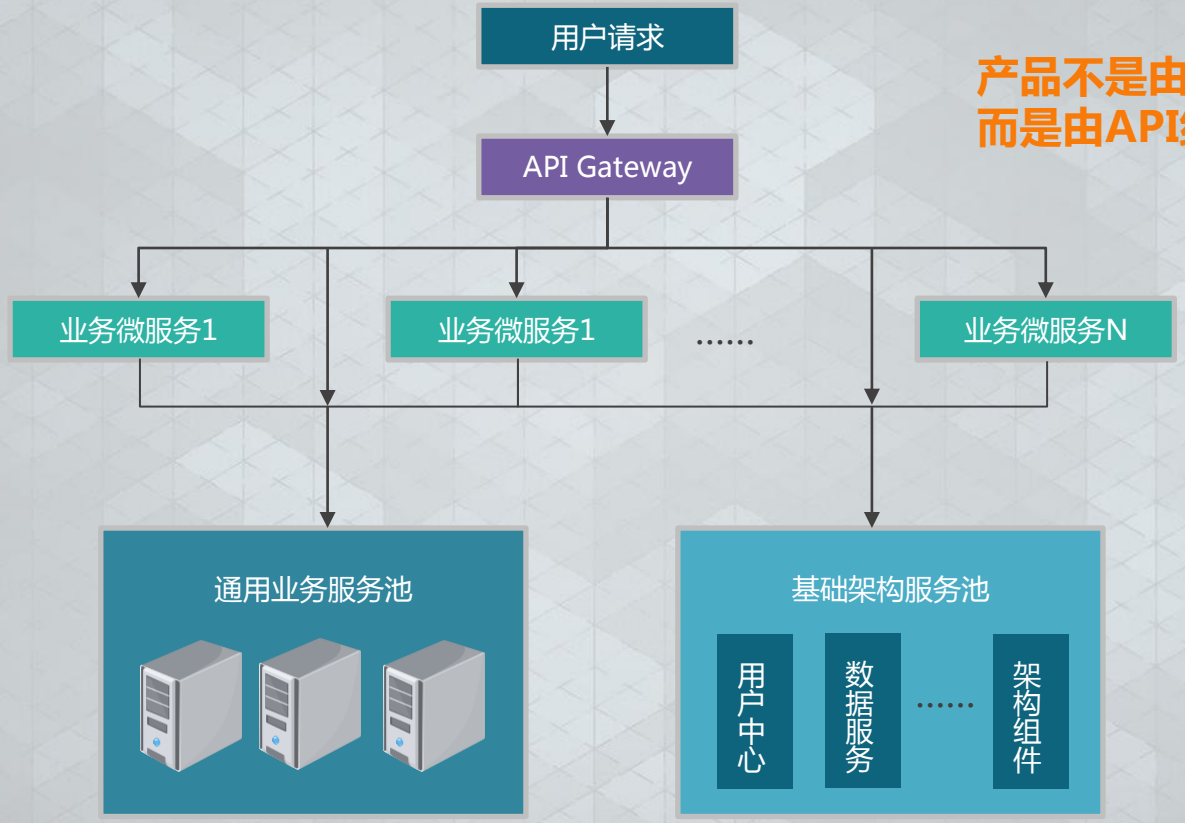
- PC/移动端
- Vue.js

- Spring Cloud
- MQ

- 业务逻辑处理
- 数据处理
- Spring全家桶

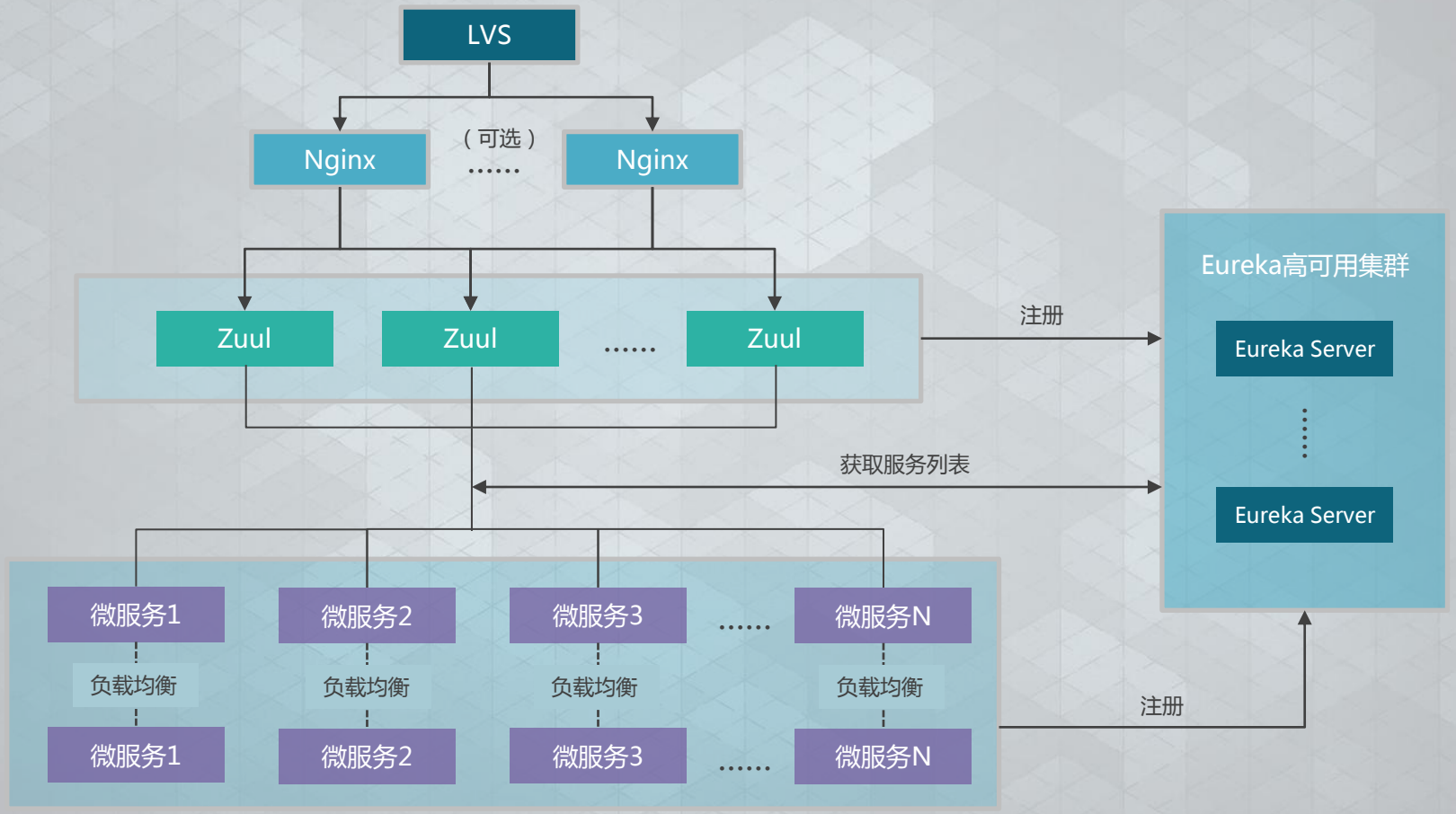
- Redis
- MySQL
- 多租户数据库设计

业务架构设计



产品不是由服务组成的
而是由API组成的

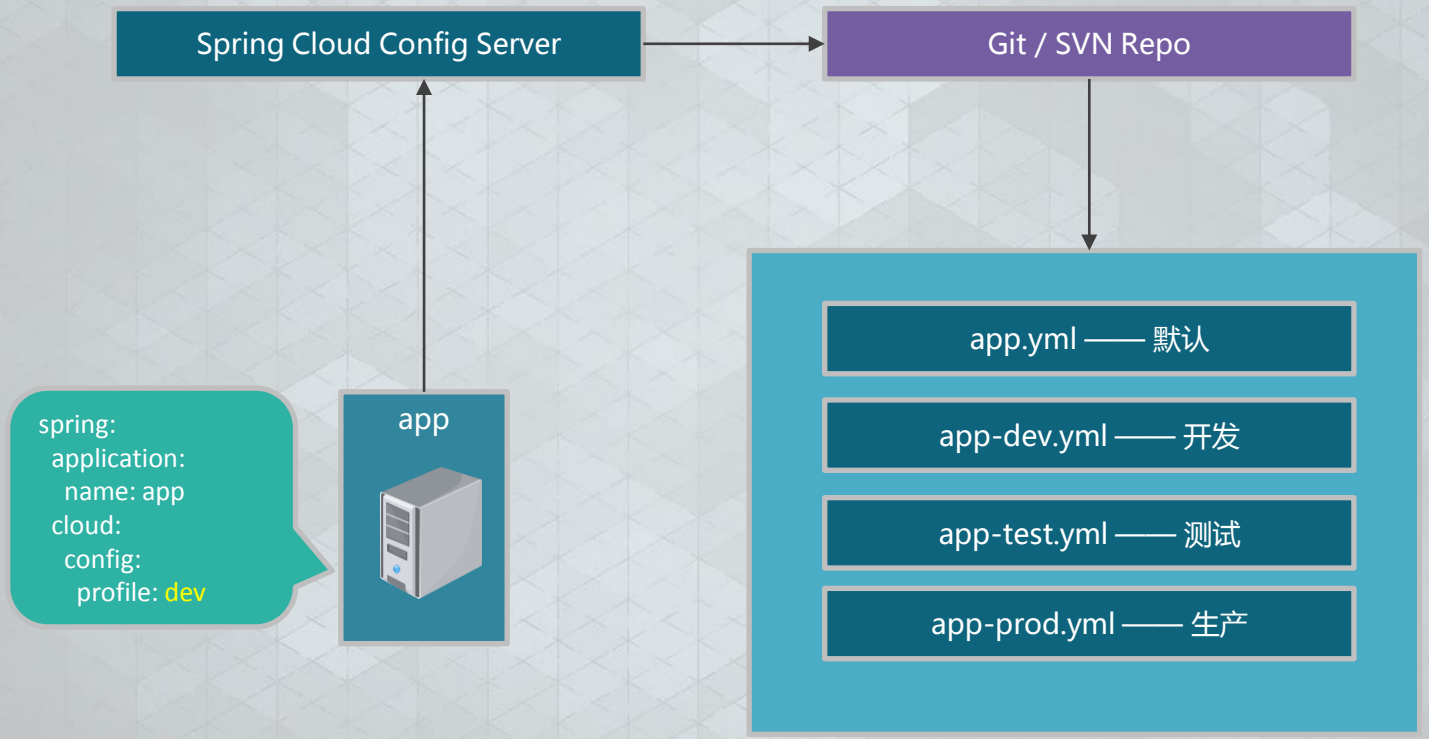
Spring Cloud 拓扑举例



实战经验分享

- 配置集中化管理
- 前后端如何更好的协作？
- 其他细节

配置集中化管理


Spring Cloud Config Server
Git / SVN Repo
app.yml —— 默认
app-dev.yml —— 开发
app-test.yml —— 测试
app-prod.yml —— 生产
app
spring:
application:
name: app
cloud:
config:
profile: dev

引用配置文件中的属性

app-dev.yml

```
my:
  test: abc
```

app-test.yml

```
my:
  test: def
```

app-prod.yml

```
my:
  test: ghi
```

一般情况：

Test.java

```
@Value("${my.test}")
private String test;
```

问题：在多处引用时比较繁琐，且修改属性名称后不好维护。

优化模式：

AppProp.java

```
@Component
public class AppProp {

    public static String TEST;

    @PostConstuct
    public void init(){
        TEST = test;
    }

    @Value("${my.test}")
    private String test;

}
```

优点：

- 注入为静态属性，可全局各处引用；
- 集中管理，方便维护。

缺点：

- 不支持@RefreshScope动态更新。

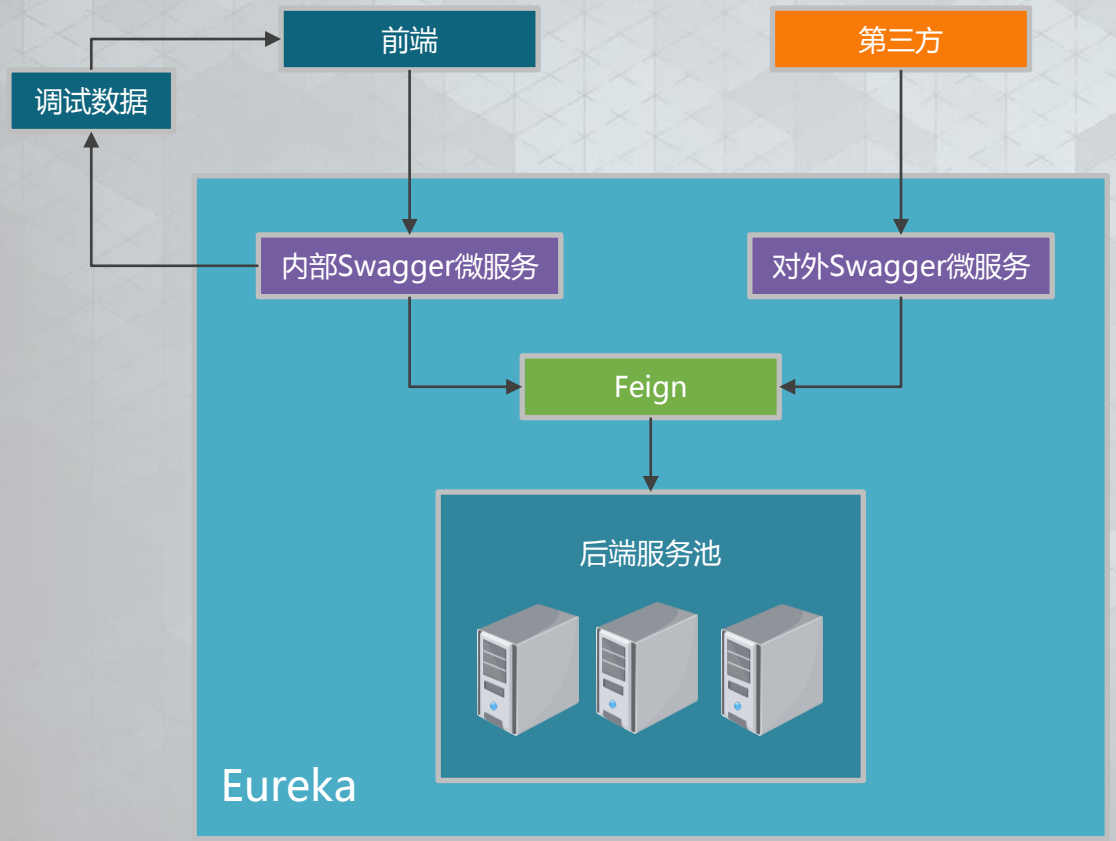
推荐把不常改动，或没有动态更新需求的属性放在这里。

前后端协作

通常使用swagger方式中存在的问题：

- 各类与业务无关的注解大量污染Controller代码，造成维护困难；
- 灵活性差，想要给特定人暴露特定接口（比如第三方）比较麻烦；
- 发布生产时需要特殊处理来关闭swagger；
- Swagger有时会与其他jar包冲突（比如springfox-swagger2.6.0会导致注册Eureka异常）。

Spring Cloud + Swagger

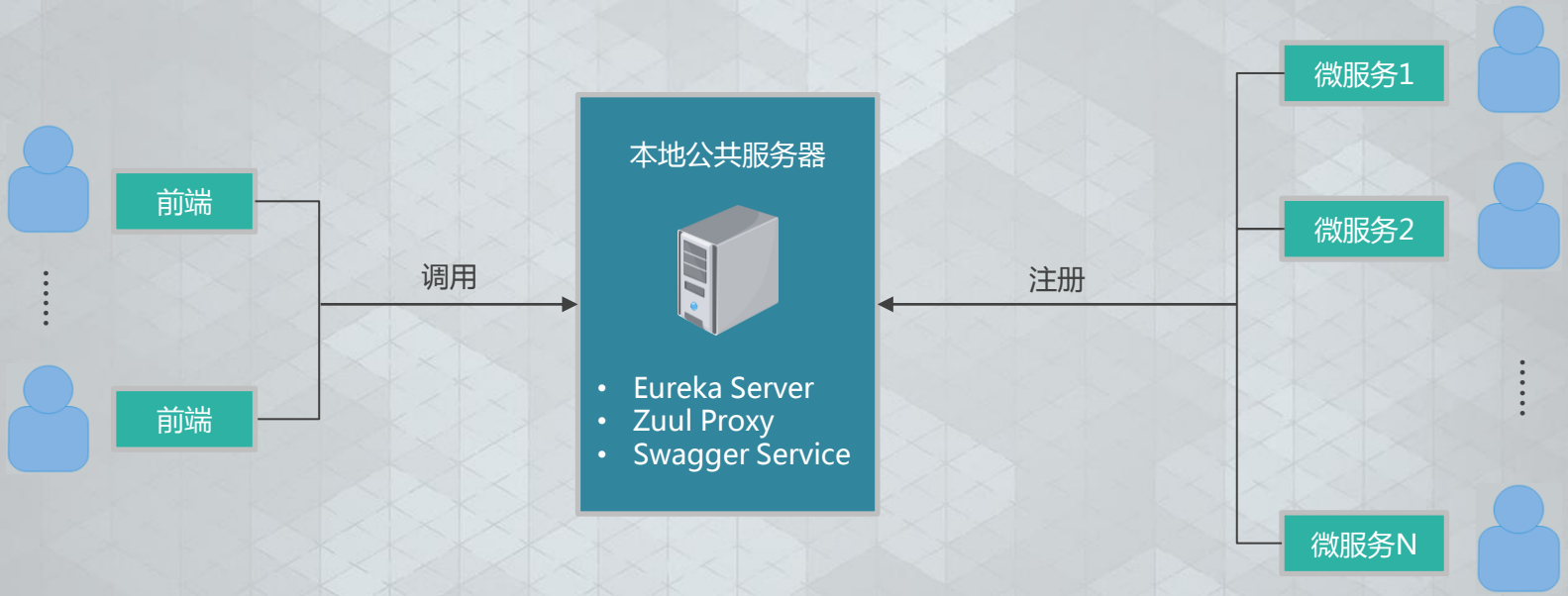


1. 开发未动，文档先行。正式开发前优先编写独立的Swagger微服务供开发人员参考，让Swagger回归文档本质；
2. 项目初期，由Swagger微服务直接返回格式化的假数据供前端调试，方便前后端并行开发；
3. 前后端联调时，前端可继续由Swagger通过Feign来调用后端服务查看数据，或直接连后端服务调试真实业务逻辑；
4. 可针对不同第三方的需求，提供不同的对外Swagger微服务让对方调试，灵活暴露接口。
5. 后端服务完全不引入任何Swagger代码，保持代码纯净，也避免了Swagger冲突，发布生产时直接关掉Swagger服务即可；

注意事项：

1. Swagger的接口路径、参数等必须与真正的业务接口保持一致，严格遵守规范，方便前端直连后端时统一修改；
2. Swagger微服务中需要有相应的VO，这类东西可以编写一次，到处复制。因此并不会增加工作量。

前后端联调


前端
前端
调用
本地公共服务器
Eureka Server
Zuul Proxy
Swagger Service
注册
微服务1
微服务2
微服务N

其他细节

01 单实例超时时间

```
hystrix:
  command:
    default:
      execution:
        isolation:
          thread:
            timeout-in-milliseconds: 15000
```

02 负载均衡超时重试

```
ribbon:
  ReadTimeout: 15000
  ConnectTimeout: 15000
  MaxAutoRetries: 0
  MaxAutoRetriesNextServer: 1
```

03 还想用XML配置？

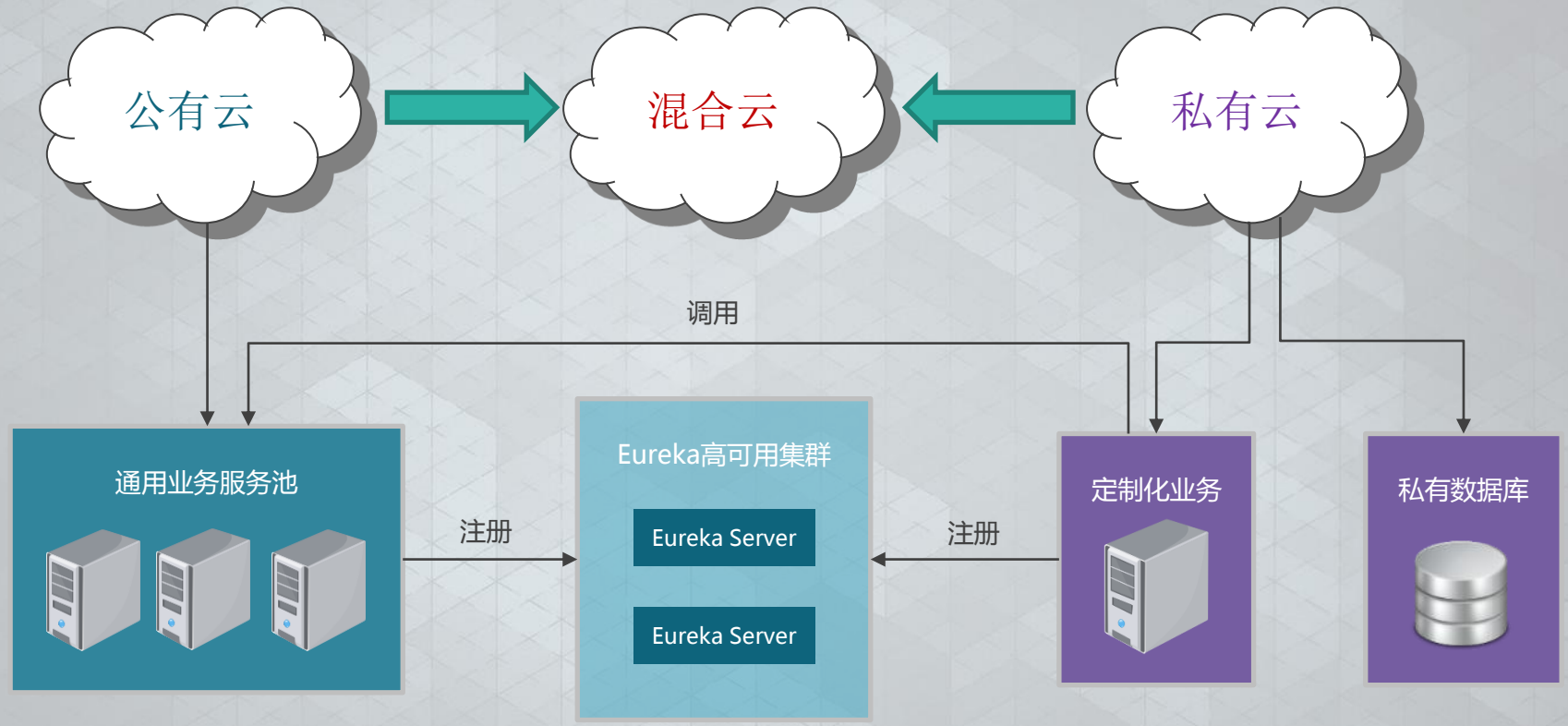
有些深入配置YAML不支持，可以使用XML来提供：
在启动类上添加注解
@ImportResource(locations={"classpath:spring/spring.xml"})

注：在XML中可直接引用YAML配置的属性，无需使用context:property-placeholder

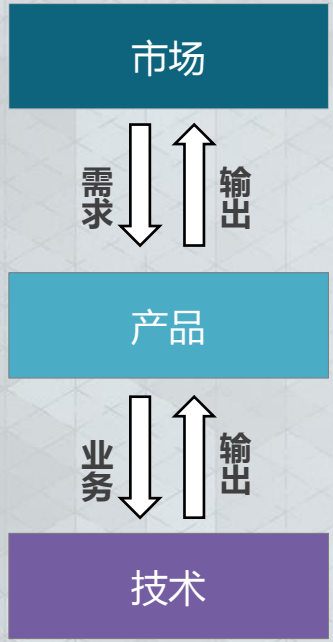
总结

- Spring Cloud + SaaS = ?
- 技术并不是全部

Spring Cloud + SaaS = ?



技术并不是全部



1. 重要性：市场 > 产品 > 技术
2. 技术是为产品服务的，产品是为市场服务的
3. 技术要稳中求新
4. 实现功能后要多关注用户体验
5. 营销策略、商业模式更为关键
6. 技术人员对产品也应有自己的理解

THANKS