Week 9-10：微服务架构

接下来我们将要探寻将这个平台改造成微服务架构的方法。

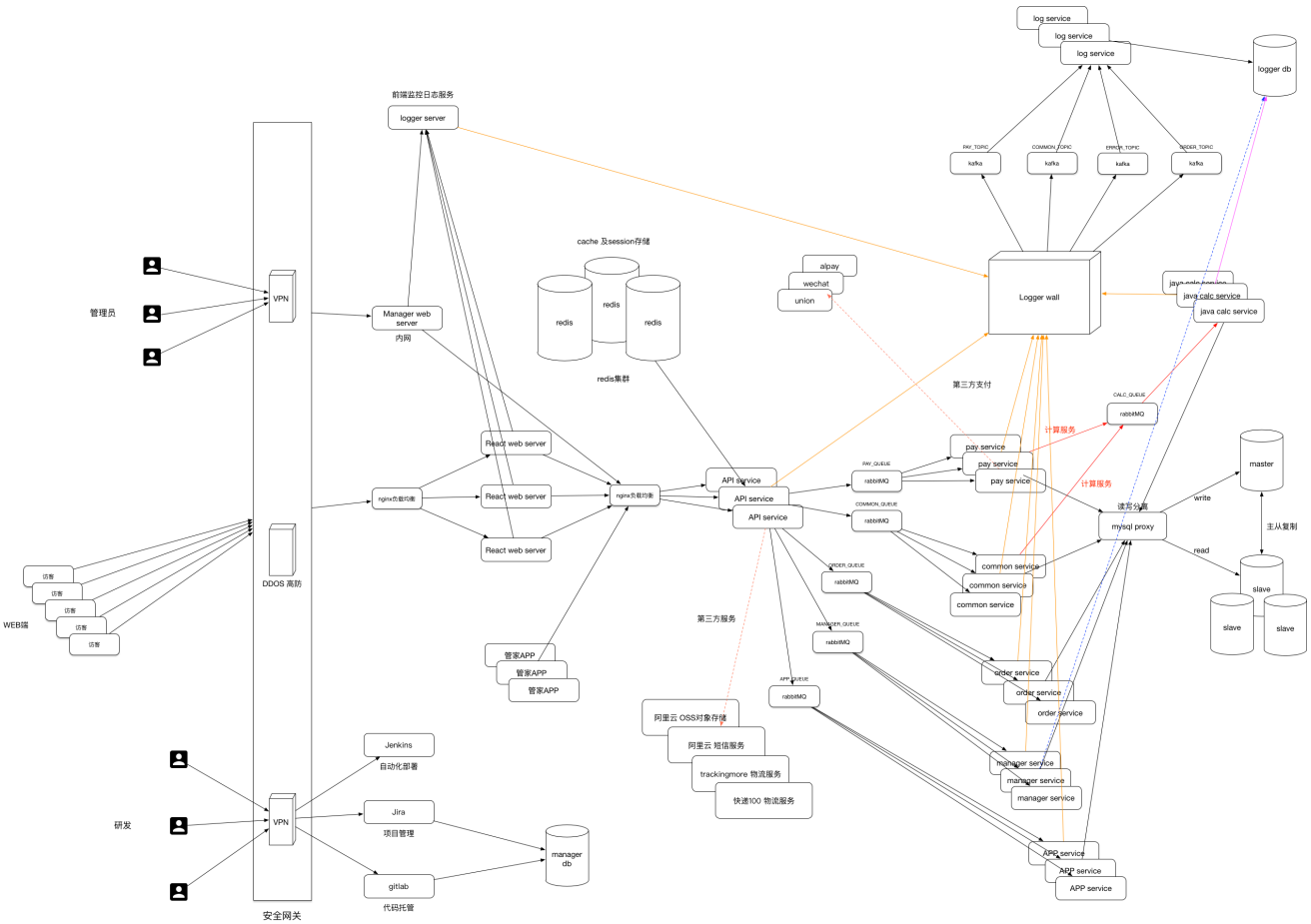
什么是微服务？

微服务适合用在团队可以独立设计、开发、运行服务的架构体系中。它允许系统中的各个服务存在技术多样性，团队可以在适合的场景使用合适的开发语言、数据库和网络协议。例如，一个团队中使用 JSON 和 HTTP REST，而另一个团队则可能使用 gRPC 和 HTTP/2 或者像 RabbitMQ 这样的消息代理。

有些场景中使用不同的数据序列化方式和协议可能收益巨大，但是需要使用我们服务的客户端可能会有不同的需求。由于存在各种各样的客户端，需要我们支持的数据格式也是多种多样，比如一个客户端可能希望拿到的数据是 XML 格式的，而另一个客户端则希望数据是 JSON。另一个你可能需要面对的问题是，可能不同服务之间存在着一些公共的逻辑(比如权限认证之类)，总不能在每个服务里都实现一遍吧？

总结：我们不想把一些支持多个客户端等相关的公用逻辑重复实现在微服务中，我们需要一个 API Gateway 来提供一个中间层来处理服务协议之间的差异，并满足特定客户端的需求。

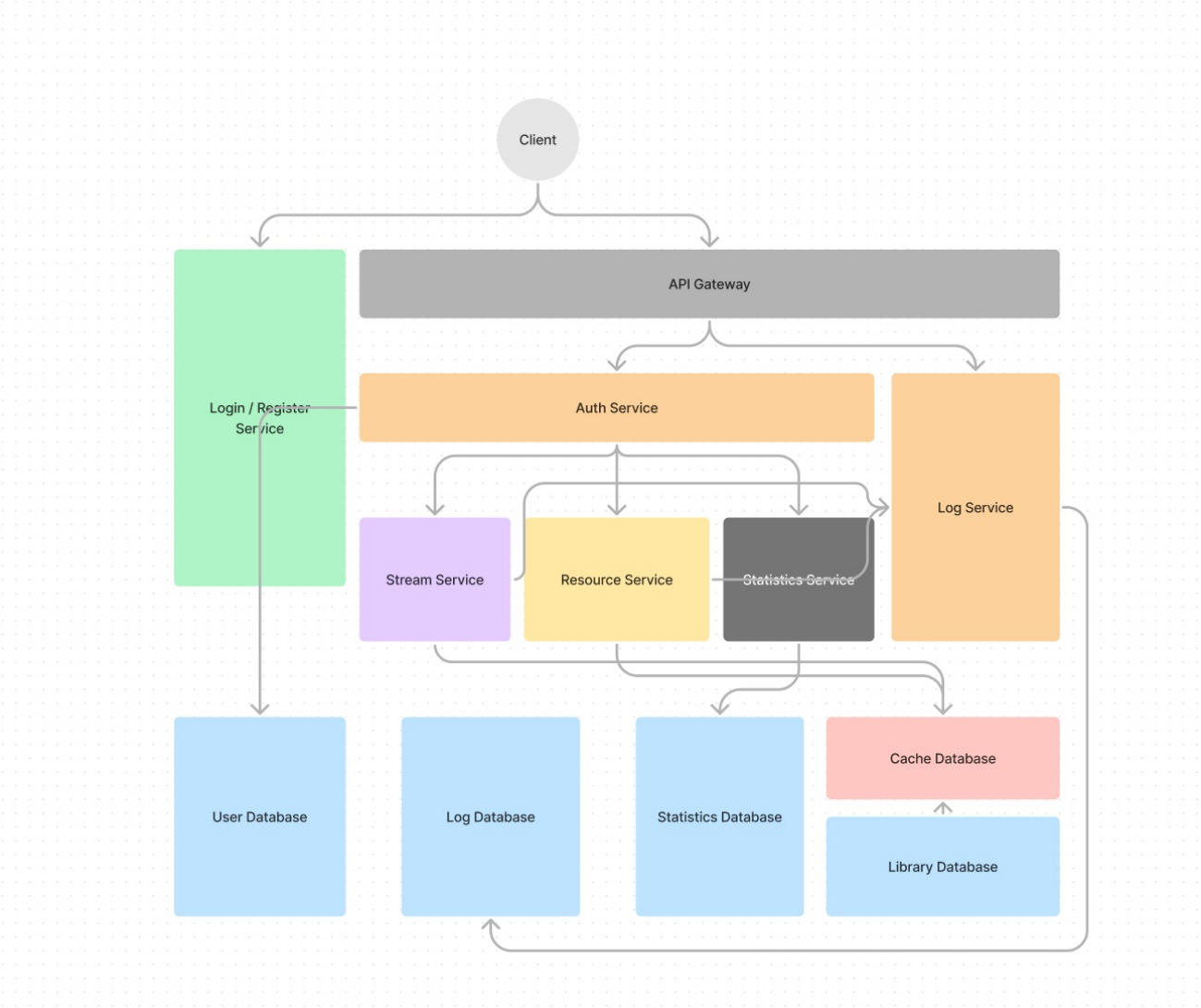
一个大型电商微服务架构参考图：



任务1

第一步，我们需要将整个后端系统重构，改成微服务形式的架构。

在这里给出一张架构参考图：



该项目最终将会完成这样的结构。当前我们能够运行的核心业务分别有：

- Login / Register Service
- Stream Service
- Resource Service

微服务的核心思想是每个业务各司其职、解耦合，单独接入或者删除一个子服务不会影响系统中其他服务的运行。

因此我们第一件事就是建立一个健壮的Api Gateway，用来路由接入的API请求。

API Gateway通常与Authentication服务高度结合，所有请求在进入子服务前就已经完成了鉴权。

在这一环节，你需要完成ytm-gateway模块，并且该模块需要承担以下功能：

1. 鉴权，解析请求携带的JWT Token，获得用户的UID和subscribe level，并且拦截无效的token
2. 路由，根据请求的来源将消息和解析后的用户信息转发给对应的子服务
3. 响应，接收到来自子服务的数据后转发回给来源
4. 自动注册，暴露一个内部端口/API接口，用于子服务的自动注册，即子服务可以向API Gateway通过Rest请求注册自己的服务名、接受的resource name、端口号、是否需要auth等信息，而无需在每次上线新的微服务时需要手动调整API Gateway的配置。其中因为音乐串流需要用到stream，所以我们的API Gateway需要专门写一些支持streamBuffer转发的逻辑。

第一阶段，我们将使用端口转发的方式在各种子服务之间通信，Gateway作为核心入口我们可以使用高性能Web服务Nginx，参考教程：https://www.nginx.com/blog/deploying-nginx-plus-as-an-api-gateway-part-1/

但是使用Nginx实现自动注册可能会有一些难度，我们推荐使用KOA自己编写自定义的业务逻辑。

API Gateway必须严格遵循HTTP Status Code的规定，比如成功返回200，试图访问不存在的接口资源时返回404等。HTTP Status Code的详细信息可以通过https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Status了解。

参考：

https://segmentfault.com/a/1190000010581422

https://cnodejs.org/topic/5a0d3c61791a31556f6ed9e1

https://medium.com/young-app-platform/top-5-opensource-nodejs-api-gateways-for-api-management-to-implement-in-2020-c6e82233ddb5

https://blog.csdn.net/wang_yu_shun/article/details/122248427

https://learnku.com/articles/30704

任务2

将Resource Service独立出来，并且初期使用Resource Service测试自动注册功能。

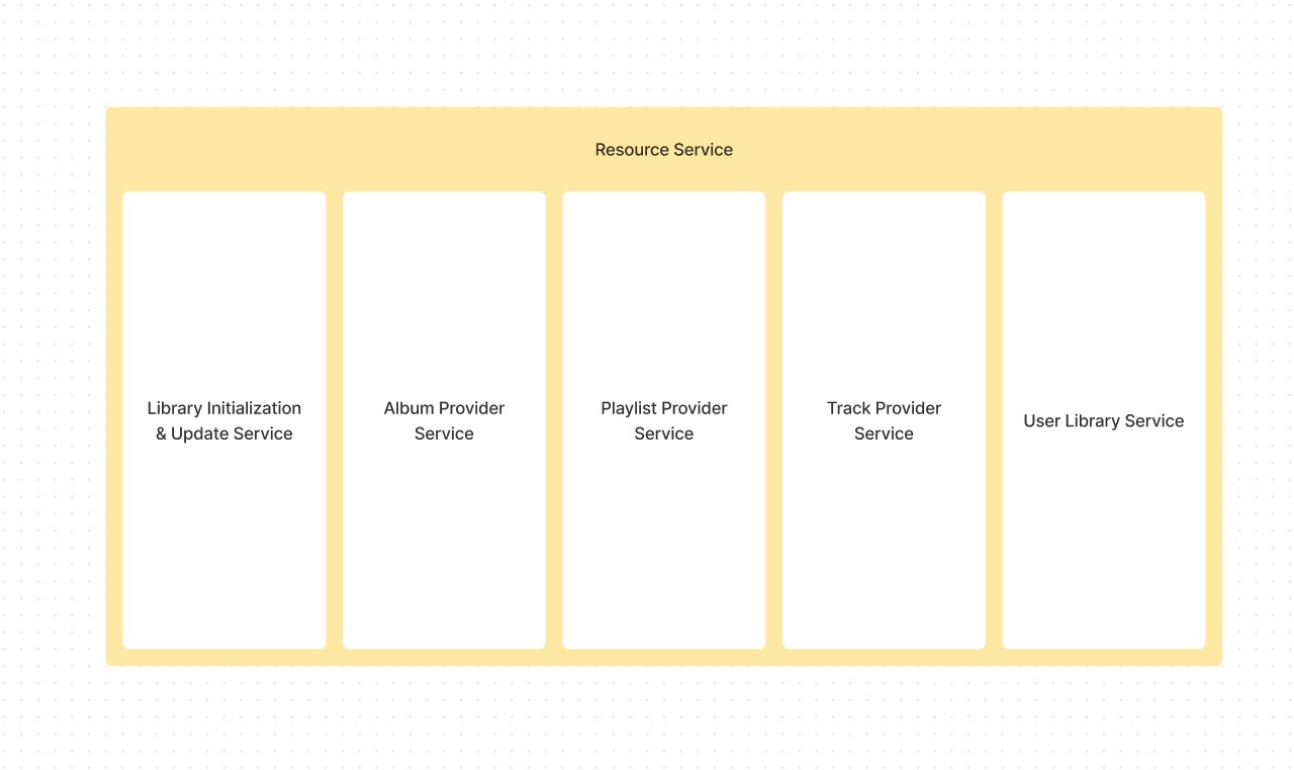
Resource Service负责提供所有静态数据，比如说首页的Explore、Library、Track、Playlist、Album、Artist等数据，由该服务与数据库对接并以规范Json返回。

该子服务所负责的权限粒度比较细，所以需要针对每一个资源考虑单独的校验，部分资源需要Auth，部分资源不需要。但向Gateway注册auth后，鉴权问题将会转变为简单的uid匹配。

一种向Gateway自动注册的模板示例：

```
{
    "name": "ytm-resource",
    "port": 35706,
    "auth": true,
    "type": "grpc",  // grpc mq stream rest
    "listen": [
        {
            "name": "album",
            "method": [
                "get",
                "post",
                "put",
                "delete"
            ]
        },
        {
            "name": "track",
            "method": [
                "get",
                "post",
                "put",
                "delete"
            ]
        },
        {
            "name": "playlist",
            "method": [
                "get",
                "post",
                "put",
                "delete"
            ]
        }
    ],
}
```

Resource Service能够进一步被颗粒化：

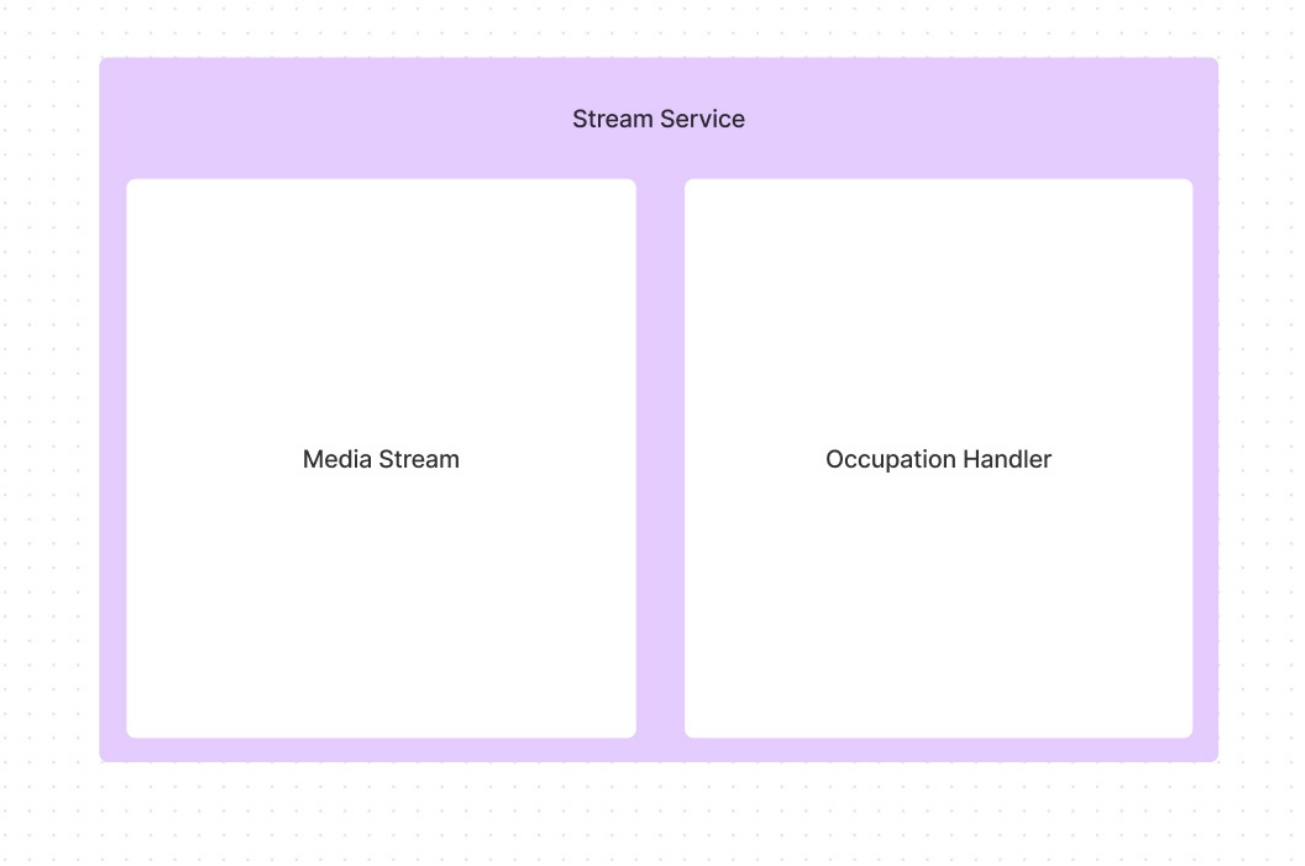


任务3

将Stream Service独立出来。

核心功能向客户端返回音频文件的streamBuffer不变，而鉴权则能够被简化为用户subscribe level的匹配，如果用户没有premium订阅，则直接拒绝串流。

API Gateway需要考虑如何转发streamBuffer。



Occupation Handler即多端挤占处理模块，该模块确保一个账户同一时间内只有一台设备正在播放音乐串流，在该项目中我们将其设定为最后播放的设备具备优先权。

该功能最简单的实现方法是通过前端向后端轮询，或者你也可以尝试更加复杂的方案，比如心跳包、Socket监听等。