微服务基础架构

修改历史：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 修改日期 | 修改内容 | 修改人 | 备注 | 版本 |
| 1 | 2018-02-27 | 初稿 | 高松 |  | 1.0.0 |
| 2 | 2018-03-31 | 追加系统形态部分，去掉持续开发部分，修改图片，修改支撑框架部分的部分描述，逻辑部署 | 高松 |  | 1.1.0 |
|  |  |  |  |  |  |

目录

[系统形态 4](#_Toc510391819)

[外部系统与微服务体系的交互形式 4](#_Toc510391820)

[微服务体系内部的交互形式 6](#_Toc510391821)

[系统结构 9](#_Toc510391822)

[微服务架构组成 9](#_Toc510391823)

[微服务系统整体结构 9](#_Toc510391824)

[微服务框架 10](#_Toc510391825)

[选型 10](#_Toc510391826)

[支撑框架 12](#_Toc510391827)

[选型和完成度 12](#_Toc510391828)

[部署逻辑结构 15](#_Toc510391829)

[设计方法论 16](#_Toc510391830)

# 系统形态

## 外部系统与微服务体系的交互形式

外部系统，不包含任何微服务体系内部服务的，以API网关为边界并且在API网关以外的所有系统，都是相对于微服务体系的外部系统。

微服务体系，有且只能包含内部微服务，以API网关为边界并且全部包含在API网关以内的所有服务。微服务体系有且只有一种入口，API网关。

如下图所示，外部系统是通过API网关服务对微服务体系进行访问。

实际选型：微服务体系选择RESTFUL接口风格，通过API网关向外部统一暴露接口。外部系统通过HTTP客户端与API网关通信并请求接口服务。



## 微服务体系内部的交互形式

#### 同步方式

微服务体系内部是由任意个微小的独立服务组成，每个服务之间都可以通过自身服务接口相互调用。

实际选型：微服务暴露的是RESTFUL设计风格的接口。所以从底层实现来说，服务之间的相互调用是通过HTTP客户端直接向指定服务的接口请求。



#### 异步方式

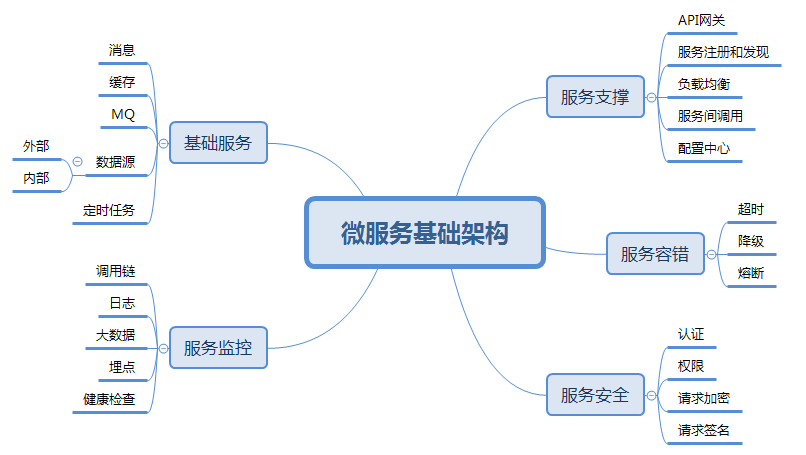
服务之间通过消息中间件进行异步消息传递，通过观察者模式异步触发接口调用。

实际选型：包装消息中间件为事件驱动模型，使用事件定义的方式进行接口调用。



# 系统结构

## 微服务架构组成



## 微服务系统整体结构



一套微服务系统可以看作一个整体，采取严进宽出原则。

注：生态系统中的数据是仅限于当前微服务系统操作，如果需要多个微服务系统共同操作的数据，数据将存储在外部公共的服务系统中。

# 微服务框架

微服务框架将提供微服务运行时的支撑服务，如服务注册和发现、负载均衡、API网关、熔断、配置中心和服务调用工具。

## 选型

微服务框架选型为Spring Boot/Cloud系列。

优点：

1. 越来越多的第三方组件发布了Spring-boot支持的版本，因此基于Spring-boot的开发速度快。
2. Spring Cloud集成了Netflix系列微服务框架，微服务相关体系组件完整度较高，同时使用门槛较低。
3. 满足性能、可伸缩性、可用性的要求。

缺点：

1. 缺少成型的第三方监控

# 支撑框架

支撑框架是在微服务框架的基础上，扩展了整个框架的运行的基本需求，如认证、权限，安全、监控等等；同时，也为了最大化支持业务开发，提供多种底层服务和第三方服务调用集合，如MQ，缓存，数据，定时任务，消息等等。

## 选型和完成度

1. 认证、权限框架

* 选型：自主研发

认证和权限框架采用了自主研发的形式，理由

1. 自主研发的框架可以更加适合业务的需求
2. 目前的主流认证和权限框架有的抽象度高，有的代码入侵大；所以在学习成本和实操成本也很大
3. 基于当前安全策略考虑

* 完成度：进行中

1. 安全框架

* 选型：自主研发

由于安全性的要求比较高，所以安全框架部分入侵到api网关中，结合网关的实现进行了加密。所以安全框架也采用了自主研发的形式。

* 完成度：完成

1. 监控框架

* 选型：待定

监控框架范围：

1. 日志监控
2. 指标监控： 系统指标，业务指标
3. 埋点
4. 消息队列

* 选型：RabbitMQ

MQ使用RabbitMQ，理由

1. Spring boot官方支持RabbitMQ
2. 文档比较多，开发难度低

注：现在不允许开发直接操作MQ，而是采用事件模型的方式使用MQ

* 完成度：完成

1. 缓存

* 选型：Redis（分布式），echcache（本地）
* 完成度：进行中

1. 数据（完成）

现在数据操作分为两个部分

1. Mysql数据库使用Mybatis进行访问，Mybatis采用了自动生成方式
2. MongoDB数据库使用Spring-data访问，将会推出简化版的使用jar，方便开发使用。
3. 定时任务

* 选型：quartz
* 完成度：完成

1. 消息

* 选型：自主研发

消息包括短信、站内信等发送信息方式。具体方式已经实现，现在正在研究整体的消息控制。

* 完成度：进行中

1. 工作流

* 选型：activiti
* 完成度：进行中

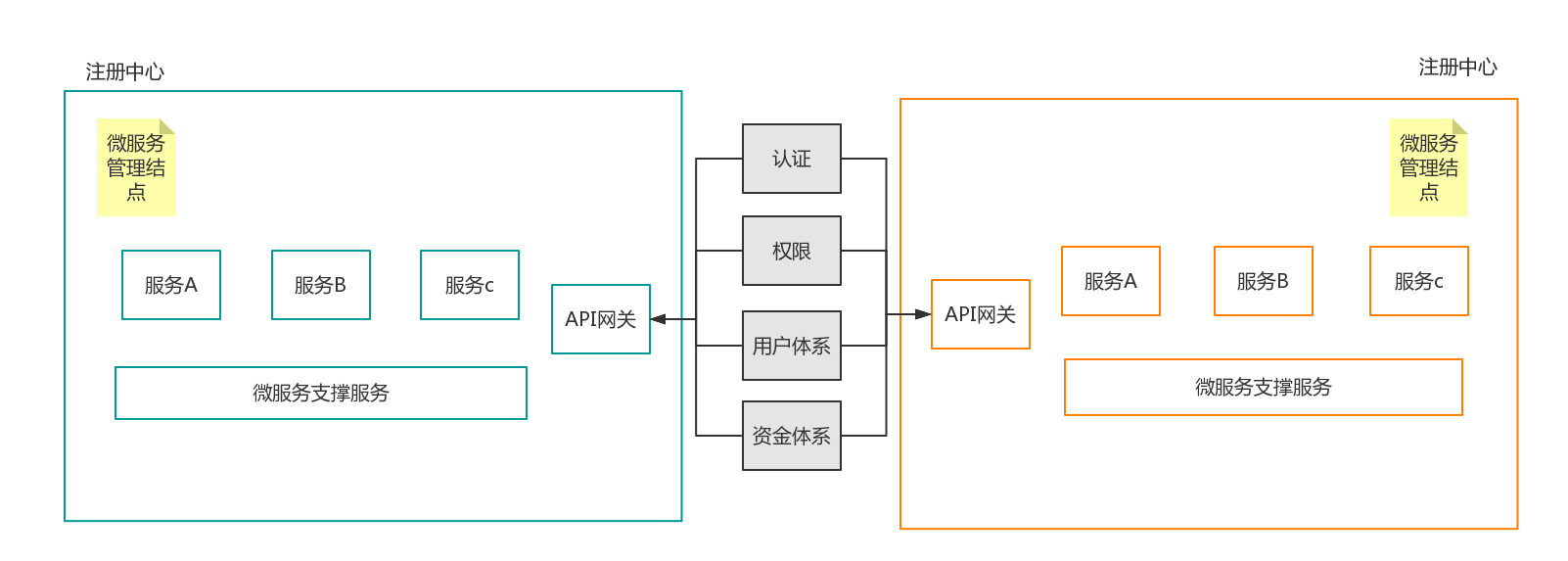
1. 规则

* 选型：待定

# 逻辑部署结构

基于spring-cloud的微服务，从逻辑方面说：

* 单个服务节点的部署，服务注册中心管理整个节点所有服务，API网关从服务注册中心获取服务地址，并反向代理。
* 多个服务节点的部署，每个服务节点内部服务的部署不变。指定共享服务在服务节点外部部署（可以独自设立注册中心）。API网关不仅从服务注册中心获取服务地址，并且会直接负载注册中心外部服务，如下图所示。



# 设计方法论

微服务，相对于传统应用开发，更加独立，自制。所以对比传统的以数据为中心的设计方法，面向模型的设计方法更加适合。

面向模型开发的优点：

1. 模型、数据、接口可以分离设计，以模型为中心，衔接数据和接口。
2. 分工更加明确，任务更加明确
3. 不再被数据绑死，数据可以以任何形式提供