SpringCloud

五大组件

Nacos 注册中心

Ribbon 负载均衡

Feign 远程调用

服务保护 sentinel

Gateway 网关

服务注册和发现

> 是啥

注册中心核心作用是服务注册和发现

常见的 **nacos** zookeeper eureka

> *eureka*

服务提供者 把服务（多个地址多台服务器）提供到注册中心　　服务：地址

消费者从注册中心拉取注册中心的对应信息 还有负载均衡 然后远程调用

服务提供者心跳续约 每30秒一次 发给注册中心 保证服务健康

> *nacos* 与*euraka*的区别

和eureka的区别在于　1.服务提供者采用临时实例 与非临时实例 两种方式　查是否存活

临时实例：心跳检测主动给注册中心查 和euraka一样

非临时实例：注册中心主动询问服务是否存活　心跳不正常 非历史实例不会被剔除

持久实例会一直存在，直到明确地被手动删除或发生某些特殊的操作（如 Nacos 服务重启，或者删除操作）。

持久实例适用于 长时间存在 的服务，或者那些不依赖于心跳机制而需要长时间存活的实例。

2.服务变更 注册中心会主动Push信息给服务消费者 更临时

3.nacos默认AP模式（高可用模式）　存在非临时实例则采用CP模式 （强一致）

4.nacos还支持配置中心

负载均衡

> *ribbon*

ribbon从注册中心拉取服务信息，返回服务列表

ribbon决定选谁 （在发起方决定 和服务器无关）

策略很多

1.简单轮询服务列表 （顺序）

2.按照权重来选择服务器，响应时间越长，权重越小

3.随机选取

4.选择并发数较低的服务器

5.重试机制 （轮询选择服务器，失效了后指定时间重试）

6.可用性敏感策略，先过滤非健康的，再选择连接数较小的

7.区域敏感策略（默认）

> 如何自定义 两种方式

实现IRule接口 配置类 (全局生效 因为配置了bean)

局部生效指定了对应的服务（配置文件的方式）

服务雪崩

服务之间互相调用 但是其中一个服务挂了 那其他服务也寄了 连锁反应

整条链路都失败了

> 熔断降级

服务降级 （部分服务）

其实就是异常了 走其他方法 （override方法） 例如提示失败

服务降级并不会完全关闭所有功能。通常是将某些非关键功能降级或延迟处理。比如，可能暂时关闭某些高负载的API接口，或者返回一些简单的默认数据，但核心功能依然能正常工作。

服务熔断 (熔断整个 服务) (如何核心功能也寄了)

Hystrix熔断 10秒内请求的失败率超过50% 那就触发熔断 之后每隔5秒重新尝试请求微服务

closed open half-open

闭路 开路 半开路

熔断=开路 一段时间之后变半开路 尝试放行一次请求 （5秒） 可以那就关闭断路器（恢复）

降级还是熔断还是看服务的重要程度 核心服务肯定不可能降级的

降级策略有三种

慢调用比例降级 在统计时长内，若慢调用比例 > 设定阈值（如 50%），且总请求数 > 最小请求数（如 30000），则熔断 N 秒（如 5s），期间所有请求直接失败。 还有慢调用的阈值

异常比例降级 在统计时长内，若异常比例 > 设定阈值（如 50%），且总请求数 > 最小请求数（如 1000），则熔断 N 秒（如 5s），期间所有请求直接失败。

异常数降级　在统计时长内，若异常数 > 设定阈值（如 100），则熔断 N 秒（如 5s），期间所有请求直接失败。

监控

问题定位

性能分析

服务关系

服务告警

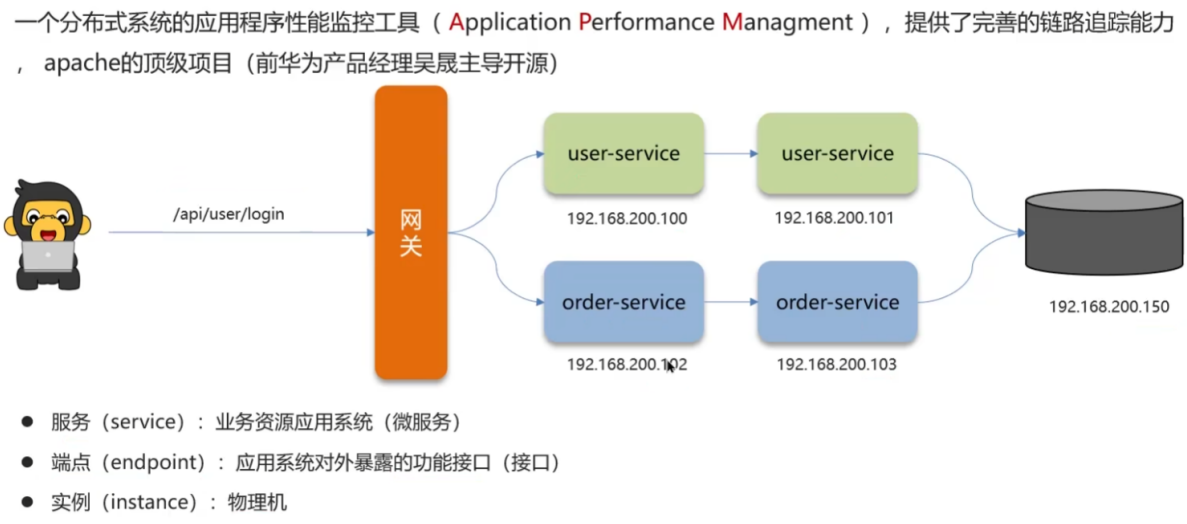
Skywalking

> *Skywalking*

服务

端点

实例



限流

并发的确大

防止恶意刷接口

Tomcat:设置最大连接数

Nginx:漏桶算法

网关，令牌桶算法

自定义拦截器

> *Ngnix*

1.控制速率 漏桶算法

漏桶以固定速率漏出请求 固定大小的桶 多余请求等待或抛弃

2.控制并发连接数

> 网关

yml配置文件 添加局部过滤器 RequestRateLimiter

定义限流对象 令牌桶每秒填充平均速率 令牌桶总容量

令牌桶算法

固定速率生成令牌，存入令牌桶，满了暂停生成

拿到令牌就处理 没拿到就阻塞或者丢弃

> 流控模式

关联模式 当 某个关键资源 **B** 承载了较大的压力时，我们希望限制 其他依赖 **B** 的资源 **A** 的流量，以防止 B 被过载。

直接：统计当前资源的请求，触发阈值时对当前资源直接限流，也是默认的模式

链路：统计从指定链路访问到本资源的请求，触发阈值时，对指定链路限流

> 流控效果

快速失败 直接抛异常

排队等待

预热模式：从小的阈值逐渐增加到最大阈值

CAP定理

C一致性 访问任意节点 数据必须一致

A可用性 必须得到响应 不是超时或拒绝

P分区 因为网络故障 节点与其他节点断联，形成独立分区

T 容错 在集群出现分区时，整个系统也要持续对外提供服务 连接恢复后再同步数据

保持高可用 不保持数据强一致 **AP**

要强一致性 **CP**

> *BASE理论*

BA 基本可用 允许损失部分可用性

S 软状态　一定时间内允许出现中间状态 临时的不一致

E　软状态结束后 最终数据一致

AP：　订单服务 扣除库存失败了 但其他服务已经提交事务　　事务协调者 ：发现有服务失败　那就删除订单 而不是回滚事务（sql)　后面再想办法

CP：等所有服务执行完成了　事务协调者才会通知服务提交事务

.

分布式事务解决方案

> *Seata框架*

TC 事务协调者

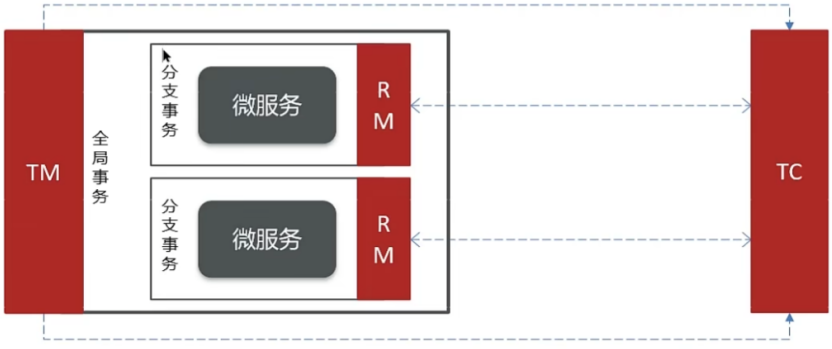
TM事务管理器

RM资源管理器

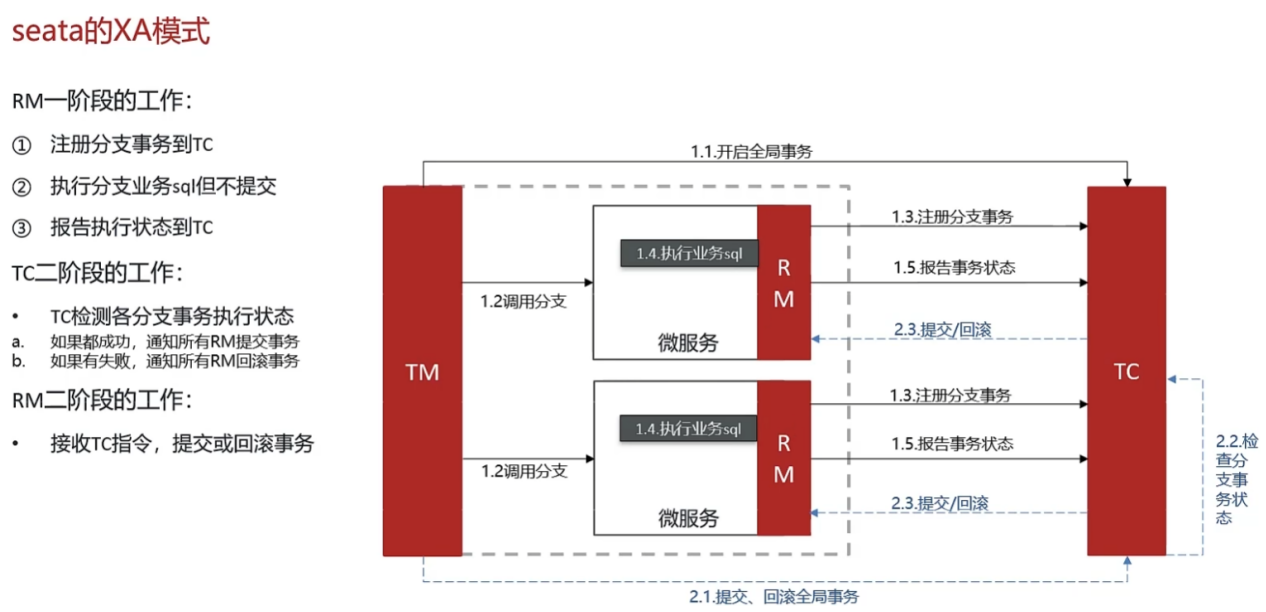
Seata架构

Seata事务管理中有三个重要的角色：

- **TC (Transaction Coordinator) - 事务协调者：**维护全局和分支事务的状态，协调全局事务提交或回滚。
- **TM (Transaction Manager) - 事务管理器：**定义全局事务的范围、开始全局事务、提交或回滚全局事务。
- **RM (Resource Manager) - 资源管理器：**管理分支事务处理的资源，与TC交谈以注册分支事务和报告分支事务的状态，并驱动分支事务提交或回滚。

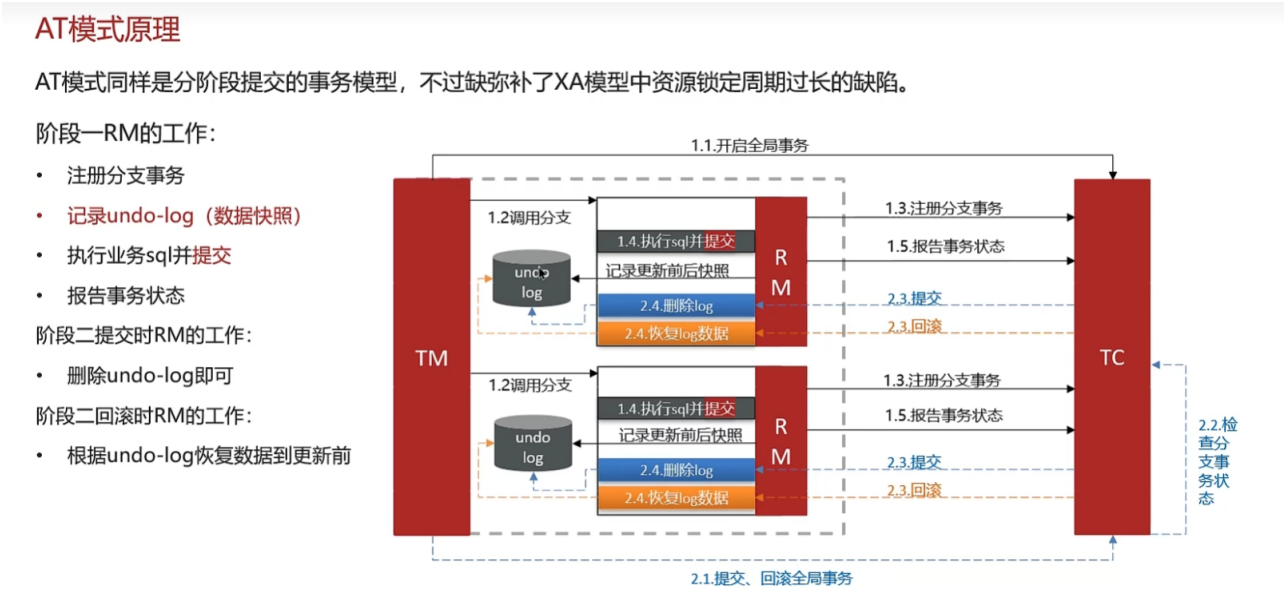


XA模式 （CP）



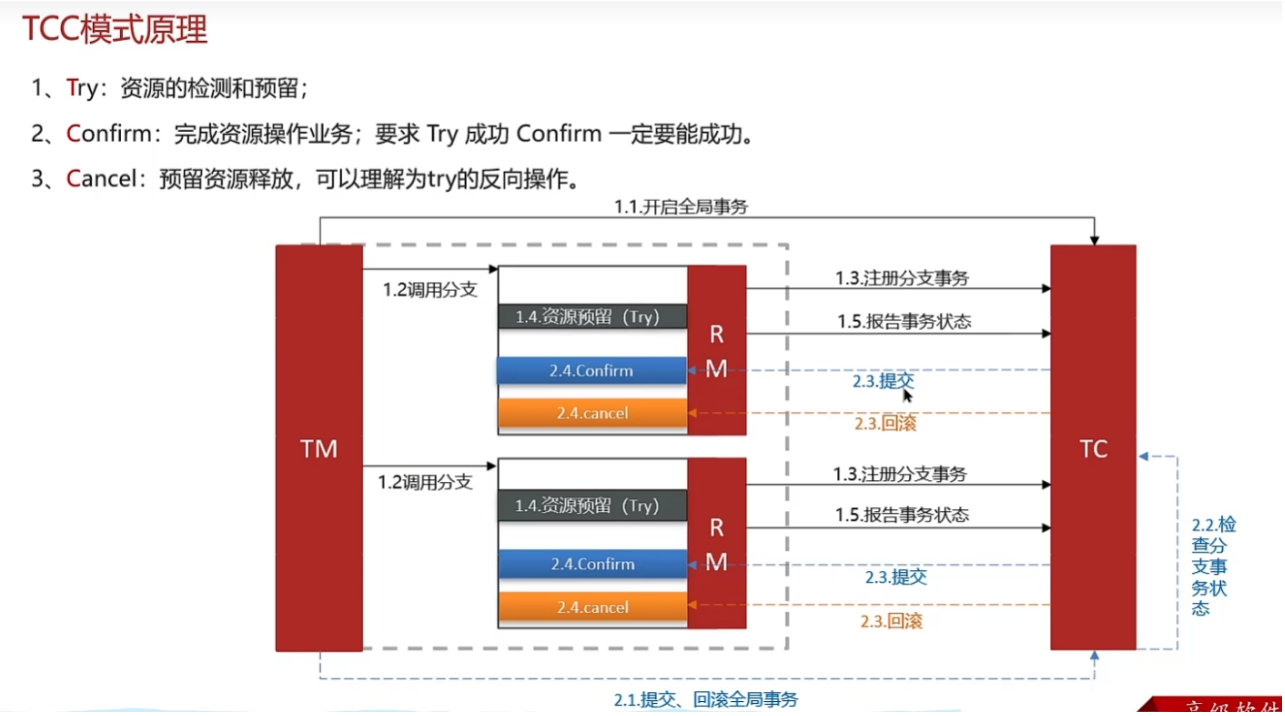
AT模式（AP）（推荐）

先交了 然后用undolog 防止出错 出错就回滚



TCC模式 （自己实现 比较难)

try confirm cancel



> *MQ*

异步

借呗 借款单发到MQ　支付宝读取到就增加余额　失败就人工解决没办法

接口幂等性

> 如何设计

多次调用方法或者接口不会改变业务状态

重复点击的场景

MQ消息重复消费

应用使用失败 超时重试

GET DELETE 是幂等

post 不是

put 以绝对值更新 是幂等的 如果是通过增量方式更新 不是幂等

三种方案：

数据库唯一索引 新增

token+redis 新增修改

分布式锁 新增修改

> *token+redis*

创建商品、提交订单

第一次请求 获取订单token 存到redis

第二次 在redis查token是否存在 存在则处理 然后删除**redis**

这种用的多

> *redisson分布式锁*

性能问题

可能不能解决？

任务调度**xxl-job**

首先要部署调度中心（一个项目） 还有需要mysql配合

它的核心结构包括 调度中心（**Admin**）、执行器（**Executor**） 和 调度流程。

调度中心提供 Web UI 进行任务管理（新增、修改、删除、调度策略）。

如果任务类型为“GLUE模式”，将会加载GLUE代码，实例化Java对象，注入依赖的Spring服务（注意：Glue代码中注入的Spring服务，必须存在与该“执行器”项目的Spring容器中），然后调用execute方法，执行任务逻辑。

GLUE模式：由于需要动态加载代码，性能略低 直接在 Web UI 上修改，不需重新部署

Bean模式：直接调用本地 Bean，性能更优 代码改动后需要重新部署应用

@XxlJob括号内的内容 `"demoJobHandler"` 是该任务的 名称

解决集群任务的重复执行问题

cron表达式定义灵活

定时任务失败了，重试和统计

任务量大，分片执行

> 路由策略

为什么要路由：很多任务 需要找一台机器

lru lfu

轮询，故障转移 分片广播 这几种比较常用

故障转移：按顺序心跳 第一个成功就选中

分片广播：所有机器都执行一次任务 同时传递分片参数

一致性哈希

> 任务执行失败怎么办

故障转移+失败重试

查看日志分析---->邮件告警

> 大数据量的任务同时都要执行

分片广播，取模

index 当前分片序号

total 总分片数 （机器数）