分布式事务专题VIP课程

朝夕教育 ZHAOXI EDU

① CAP定理解读，强一致/弱一致/最终一致

② 强一致性 2PC/3PC案例

③ TCC解析，幂等性理解和设计

④ 本地消息表分布式事务流程和组件部署

⑤ dotnetCore.CAP框架解读

⑥ 分布式环境搭建，SQLServer+RabbitMQ+MongoDB

⑦ 基于SQLServer+RabbitMQ+MongoDB生产者消费者

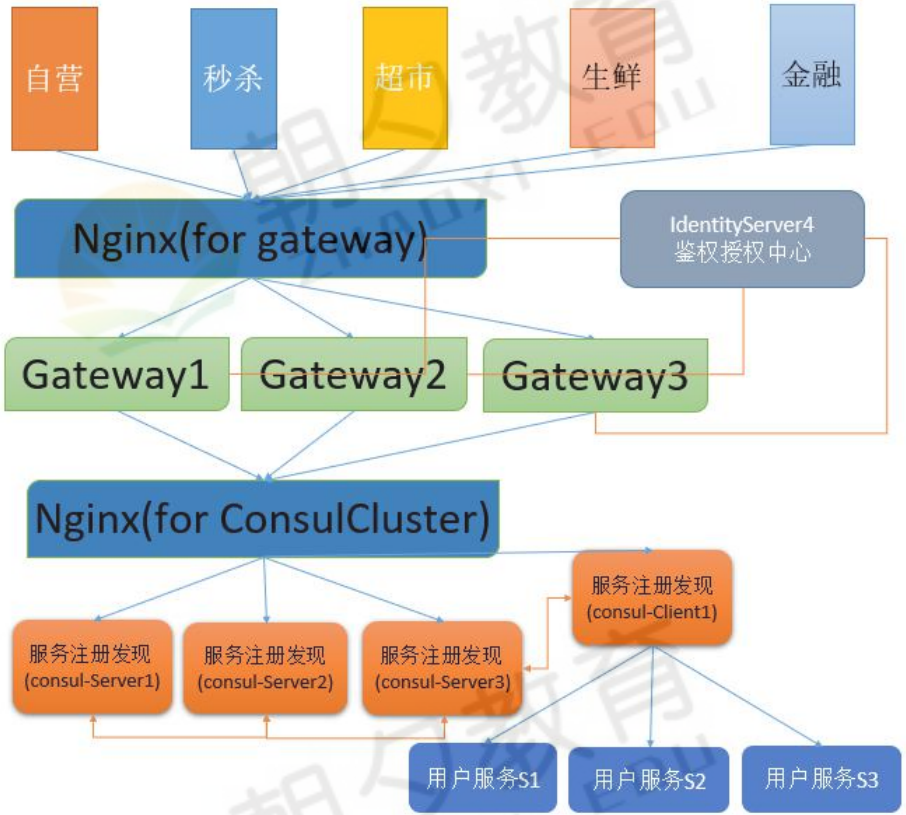
⑧ Consul监听可视化

朝夕教育 ZHAOXI EDU

微服务架构图

微服务架构，
without skywalking 和ELK

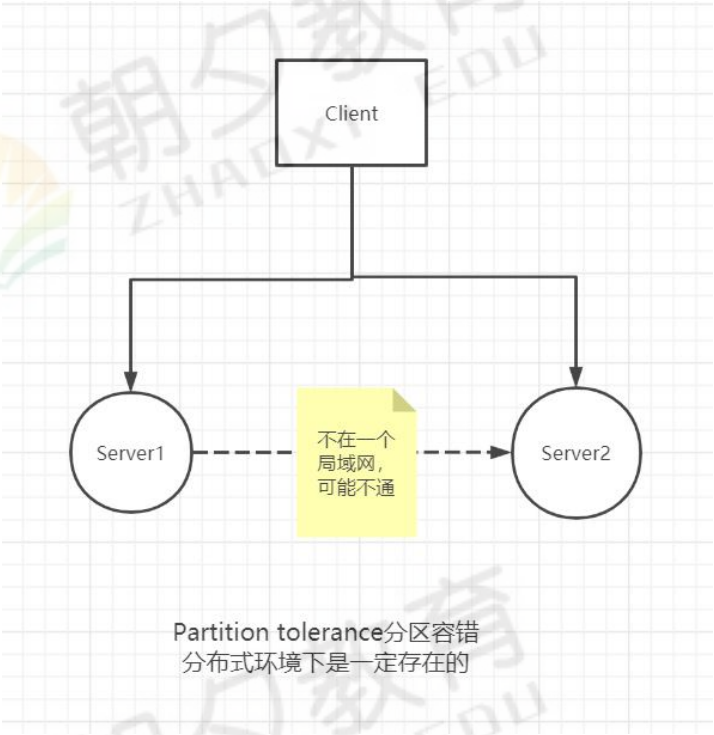
processon架构图



分布式的代价-Partition tolerance

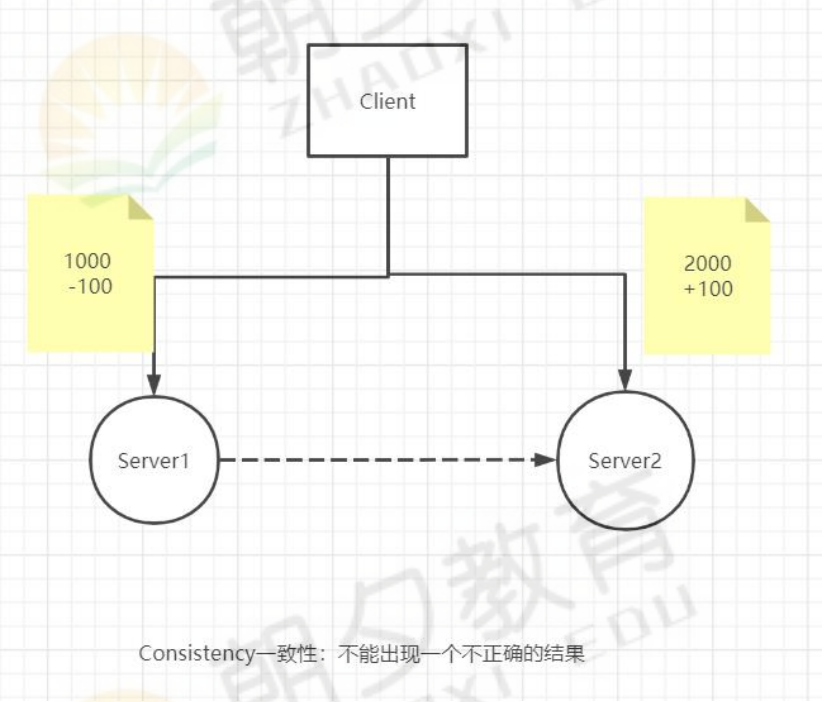
分布式环境下，服务器之间的通信，可能是不靠谱，这种情况无法避免

分区容错，一定存在



Consistency

一致性：数据得是正确的，增删改后能查出来



Consistency一致性：不能出现一个不正确的结果

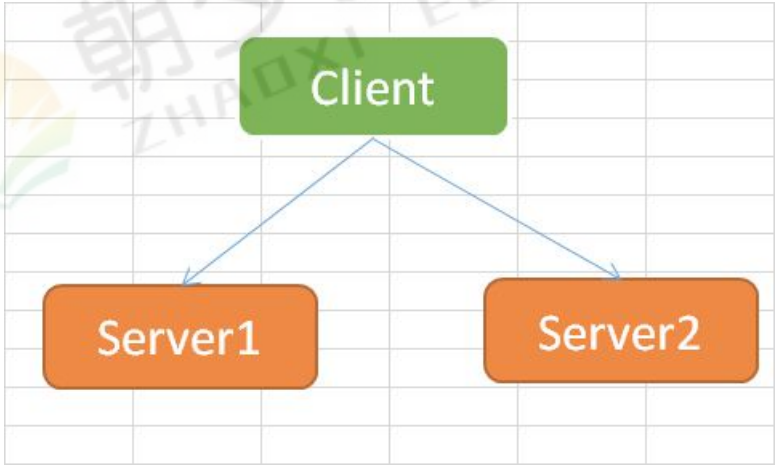
Availability

可用性：收到请求就必须响应，不能阻塞

朝夕教育 ZHAOXI EDU

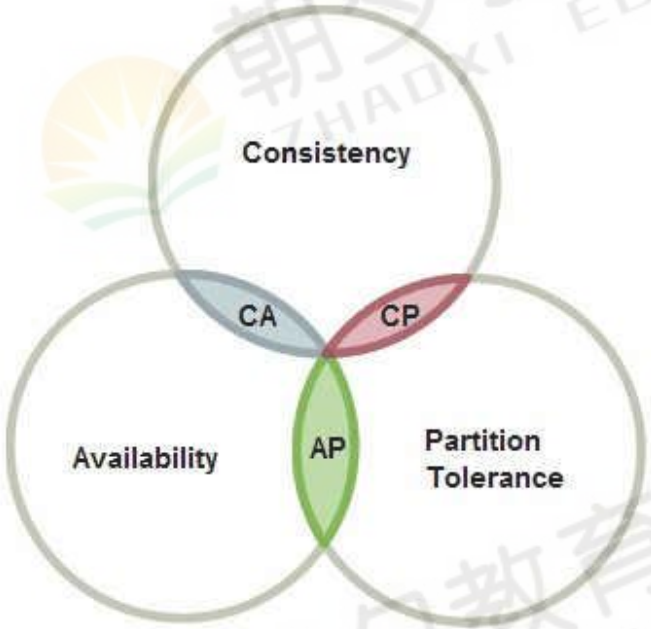
CAP定理

Consistency 一致性
Availability 可用性
Partition tolerance 分区容错
分布式系统下，网络出错是必然存在的---也就是不可靠的
在分区容错一定出现的情况，C和A是不能同时满足的



CAP是不能同时满足的！

Consistency 和 Availability怎么选



一致性和可用性，不能同时满足，要什么？

CP重要，一致性最重要了，数据不能错
银行—交易数据

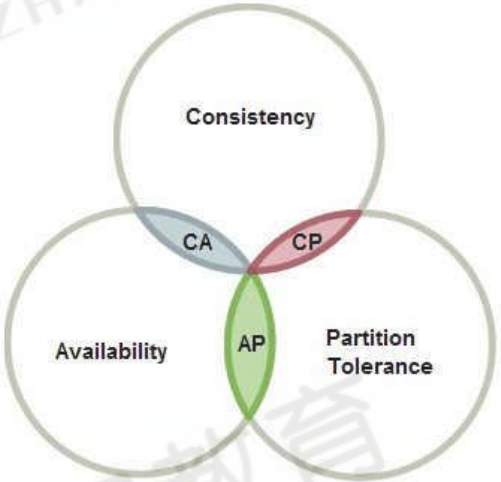
AP重要，可用性最重要了，系统的可用性，
尤其是分布式----微服务，可用性尤为重要，
没有可用性是跑不起来

多种一致性

强一致性—任意时刻数据都是一致的
2PC 3PC

弱一致性---允许某一时刻不一致，承诺在一定时间内变成一致的
Try-Confirm-Cancel 代码层面

最终一致性—允许数据不一致，但是最终最终，数据还是得一致的　业务层面



强一致性 2PC

2PC(two-phase commit protocol)
协议：规范，设计，流程

1 性能问题
2 单点故障—事务管理器
3 消息丢失问题

只是解决小范围，或者强制要求一致性，
所以就没有可用性

DTC实现

.NET Framework 下MSDTC实现
Distributed Transaction Coordinator
（演示的是单机---局域网需要配置）

.NET Core不支持—Linux---没有
Evantual2PC（不可靠）

3PC

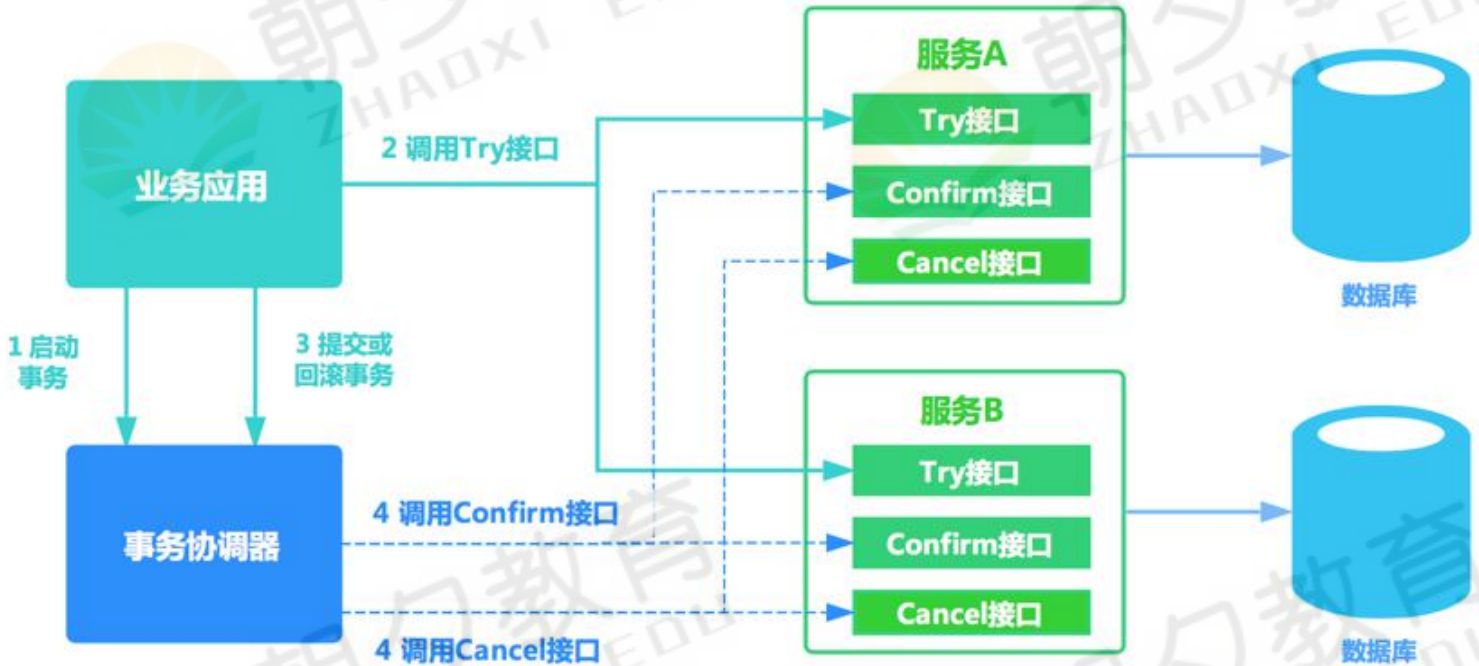
3PC(three-phase commit protocol)
Ready-GO！
CanCommit—PreCommit--DoCommit

3PC只是数据库可以自动提交
但是其他还在的，

1　性能问题
2　单点故障—事务管理器
3　消息丢失问题

只是解决小范围，或者强制要求一致性

TCC （Try-Confirm-Cancel）



TCC的注意点

1 分段设计：try成功 就一定能Confirm
2 允许空回滚:重复cancel不能错
3 悬挂控制：try阻塞，先cancel，保证数据正确
4 幂等控制：TCC多少次，结果不变
5 可见控制:值的展示
6 并发访问控制

主要用在银行、阿里---钱必须保障-不能阻塞—设计负责，开发工作重
ByteTCC、Himly、TCC-transaction事务管理器—这些都是Java的 .NET没有—所以解决方案也没有—除非自己写

朝夕教育 ZHAOXI EDU

幂等性

对同一个系统，使用同样的条件，一次请求和重复的多次请求对系统资源的影响是一致的。(网页提交按钮，点1次 跟10次结果一样)

场景：支付—页面修改信息---订单减库存

朝夕教育 ZHAOXI EDU

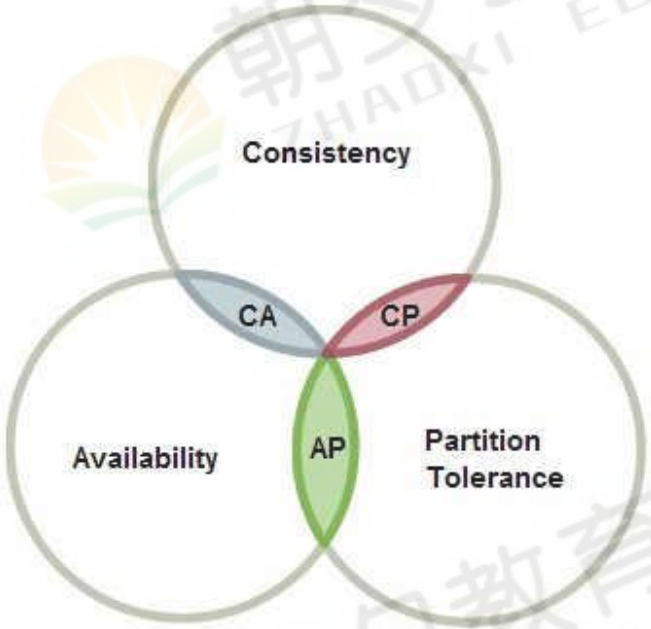
幂等性设计

1 MVCC多版本并发控制—乐观锁---数据库更新时带上版本号—跟新+1 条件必带version-----id + version

2 去重表---请求带个guid---操作前校验下guid---点赞—100赞-不能重复—文章id+用户id+唯一索引

3 Token机制---每次操作都带个唯一id，请求来了先检测再执行

最终一致性-BASE理论



Base理论：

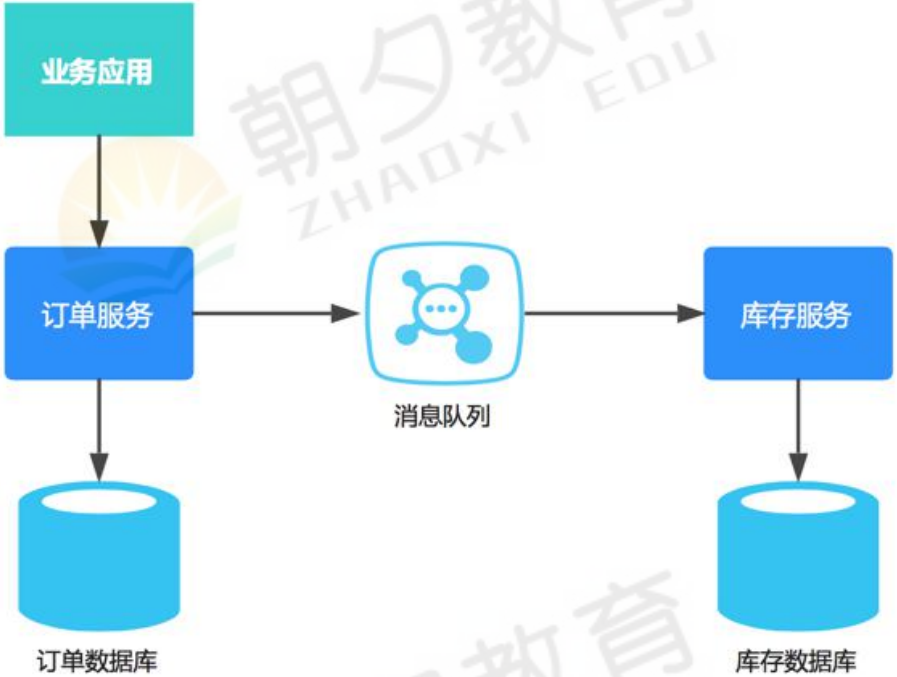
1. Basically Available(基本可用)
2. （最终一致性）
3. Soft state（软状态）
4. Eventually consistent

微服务架构里面，可用性是最重要的
思想是最重要，指引方向

本地消息表分布式事务

MQ分布式事务--本地消息表--基于消息的一致性

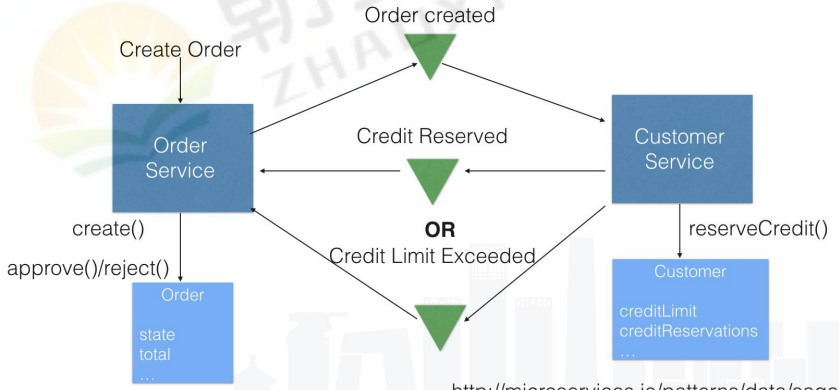
1. 上游投递消息
2. 下游获取消息
3. 上游投递稳定性
4. 下游接受稳定性



其他

比如Saga

分布式Saga实现



http://microservices.io/patterns/data/saga.html

http://www.axonframework.org/docs/2.0/sagas.html

The End

获取更多资源，对话微软MVP，微信扫一扫！

微信公众号

助教小姐姐

助教小仙女

朝夕教育
ZHAOXI EDU

THANK YOU

开发进阶，蜕变架构，升职加薪，只争朝夕！

Eleven