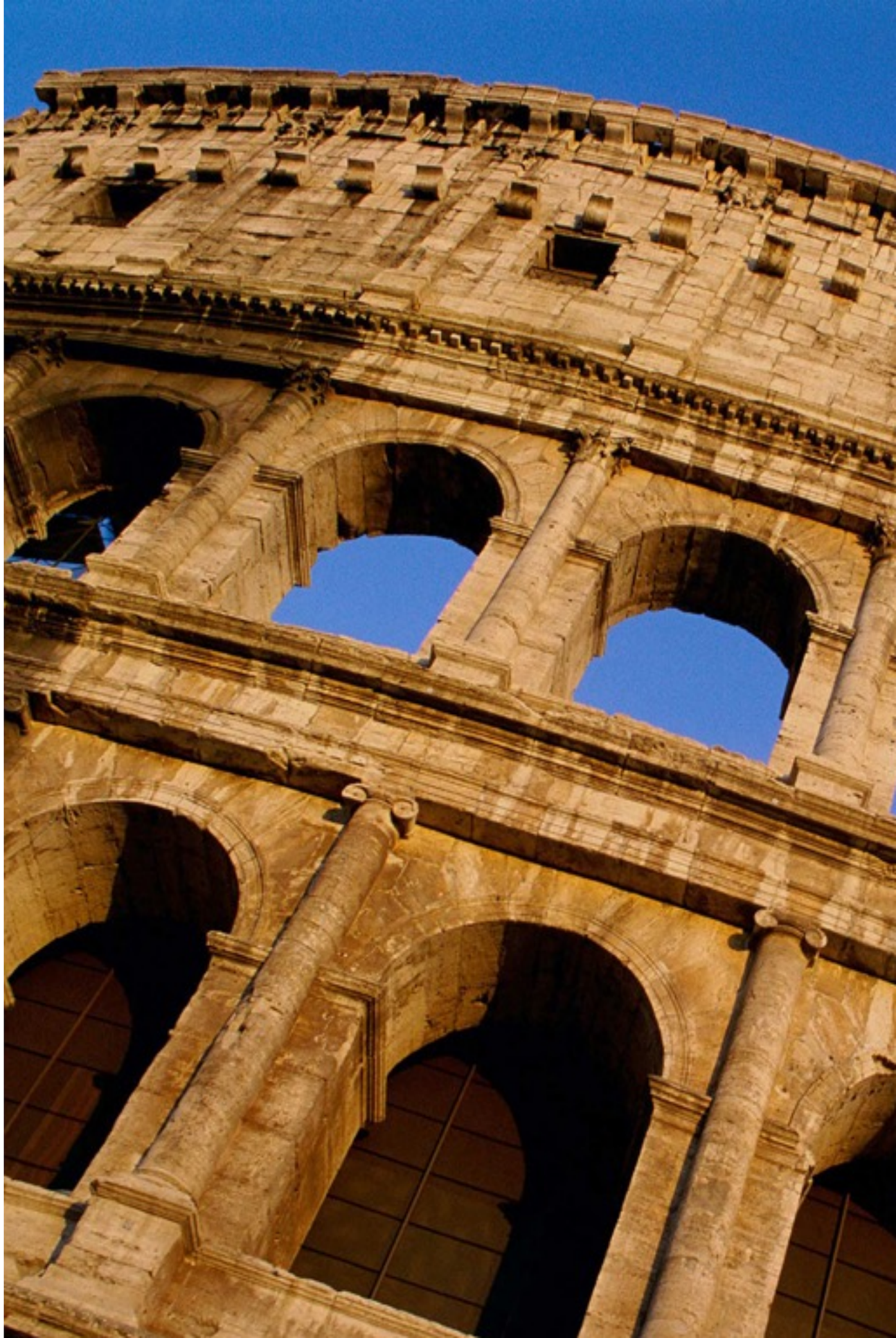
zPaaS平台产品介绍

张军勇

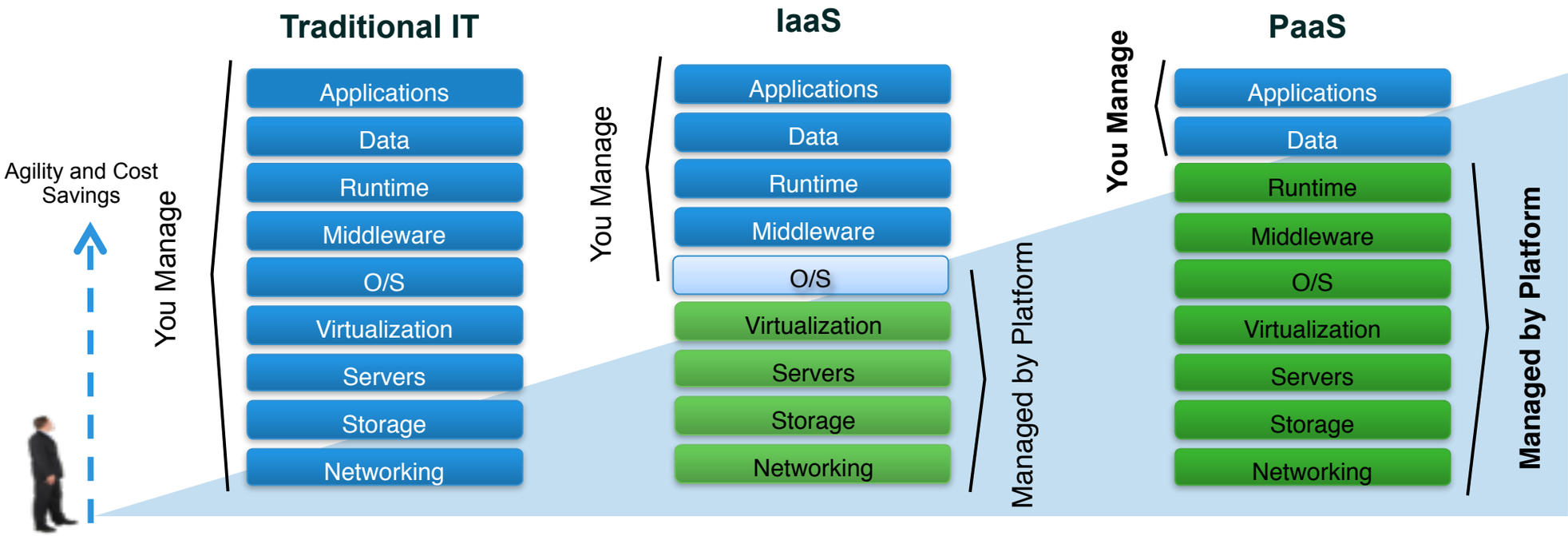
去IOE、互联网浪潮以及带来的挑战

- 去IOE：
  - I：以IBM为代表的小型机服务器
  - O：以Oracle为代表的数据库
  - E：以EMC为代表的存储
  - 去IOE就是抛弃使用的昂贵的小型机、数据库和存储，使用相对比较廉价的开源产品、PC服务器和自带存储
- 互联网架构
  - 采用各种开源产品及框架，使用价格低廉的PC服务器和自带存储
  - 需要解决大数据量、大访问量、弹性扩展能力、高可用性等问题
- 挑战：在使用各种开源产品、PC服务器和自带存储的情况下
  - 大数据量存储及大并发量访问如何解决?
  - 高可用性?
  - 弹性扩展能力?
  - 快速故障恢复?
  - 快速部署能力?

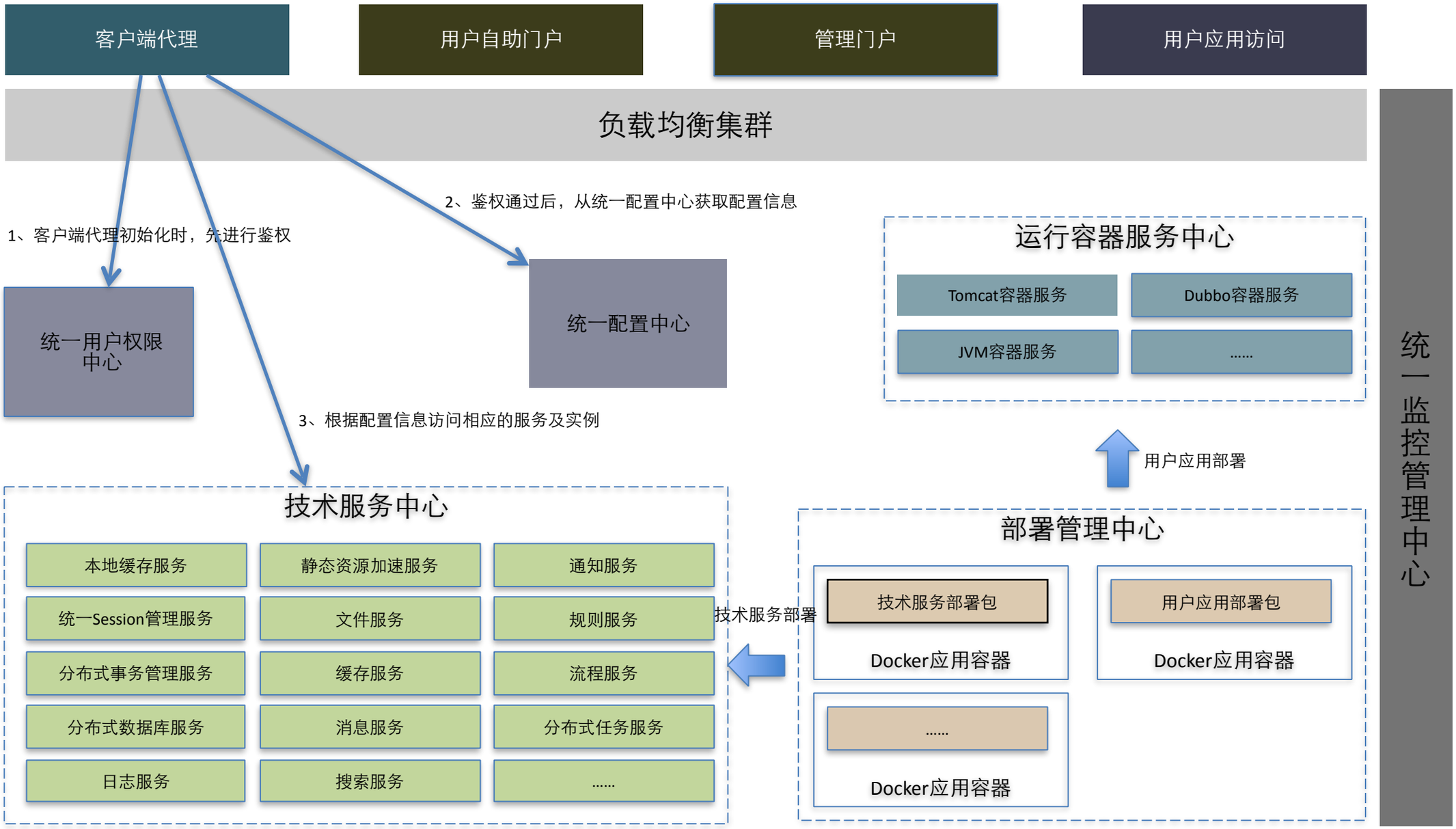
zPaaS平台能做什么



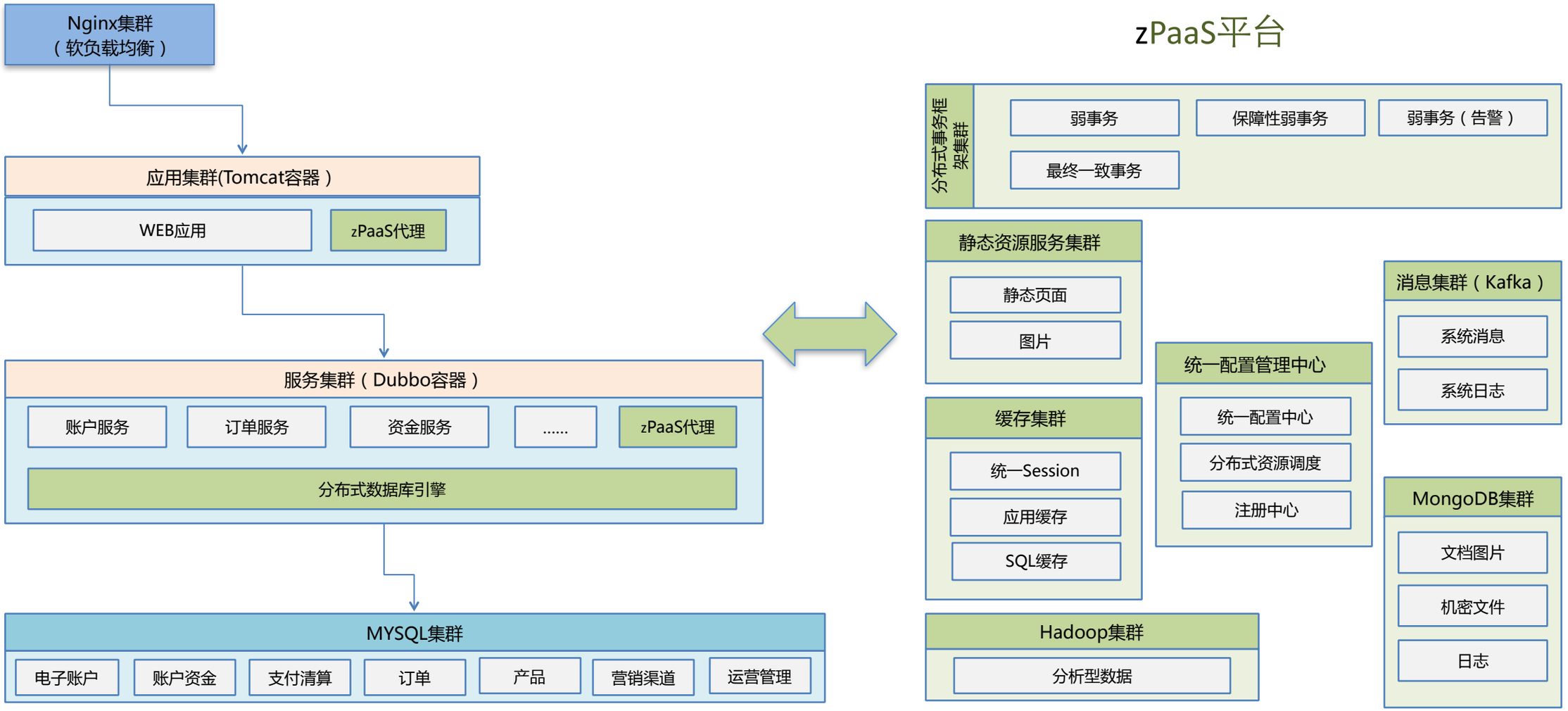
PaaS平台能做什么


Traditional IT
IaaS
PaaS
Agility and Cost Savings
You Manage
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
Managed by Platform

zPaaS平台逻辑架构



基于zPaaS平台应用的典型架构



基于zPaaS平台建设的系统具备哪些能力

- 应用基于zPaaS平台建设
  - 平台提供各种技术能力及平台能力
  - 使应用开发者可以专注于业务能力的开发，而不用关注其他技术能力及平台能力
- 大数据量及高并发处理能力
  - WEB层采用无状态多实例的部署模式，并通过负载均衡设备进行分发讲求，具备高并发处理能力
  - 服务层采用zookeeper及dubbo服务框架，使服务层可以启动任意数量的实例，具备高并发处理能力
  - 数据存储层采用分布式的关系型数据库平台以及MongoDB集群，具备大数据量存储及高并发处理能力
- 高可用性
  - Web层彩负载均衡设备进行请求分发以及统一Session服务，个体web实例的故障不会影响整体web服务的可用性
  - 服务层采用能够自注册和自发现的zookeeper及dubbo服务框架，个体dubbo实例的故障不会影响整体web服务的可用性
  - 分布式关第数据库引擎以及准实时同步工具的使用，当某个主库发生故障时，可以通过平台提供的实时更改配置功能，将备库提升为主库，继续提供服务

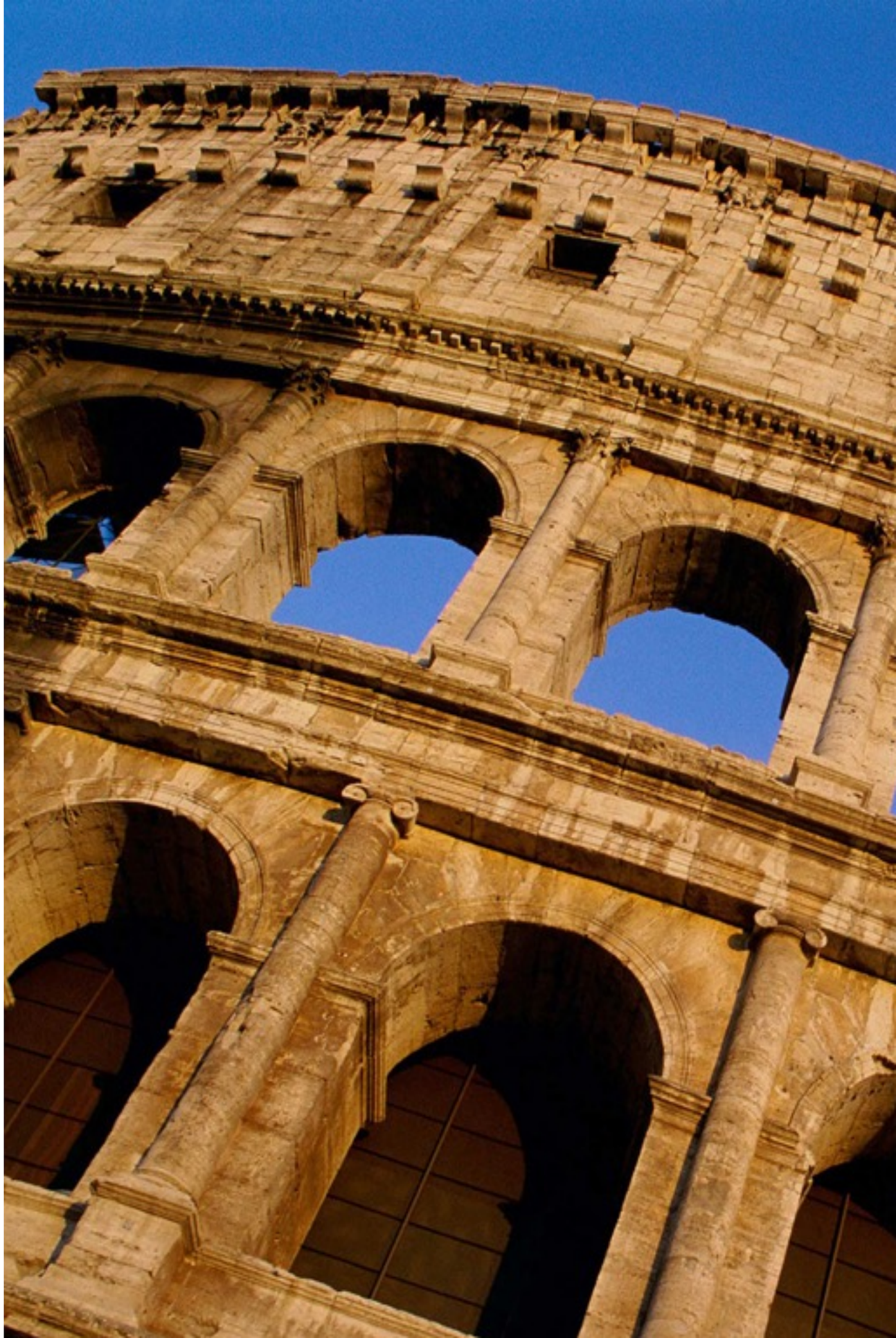
基于zPaaS平台建设的系统具备哪些能力-续

- 弹性扩展能力
  - 通过负载均衡设备以及统一Session服务使用WEB服务具备弹性扩展能力
  - 通过zookeeper及dubbo服务框架使应用服务模块具备弹性扩展能力
  - 通过分布式关系型数据库平台以及MongDB集群，使存储层具备弹性扩展能力
  - zPaaS平台的所有组件都采用可扩展的集群模式，使zPaaS平台自身具备弹性扩展能力
- 快速故障恢复能力
  - 通过zPaaS平台提供的实时配置变更功能，当有服务实例发生故障时，能够实时进行切换，如将备机提升为主机
- 快速部署能力
  - 使用容器化的部署方式，能够将应用快速复制部署到很多服务器上
- 智能运维
  - 实时故障监控及告警
  - 用户自助管理门户
  - 管理员门户

zPaaS如何实现高可用及快速故障恢复

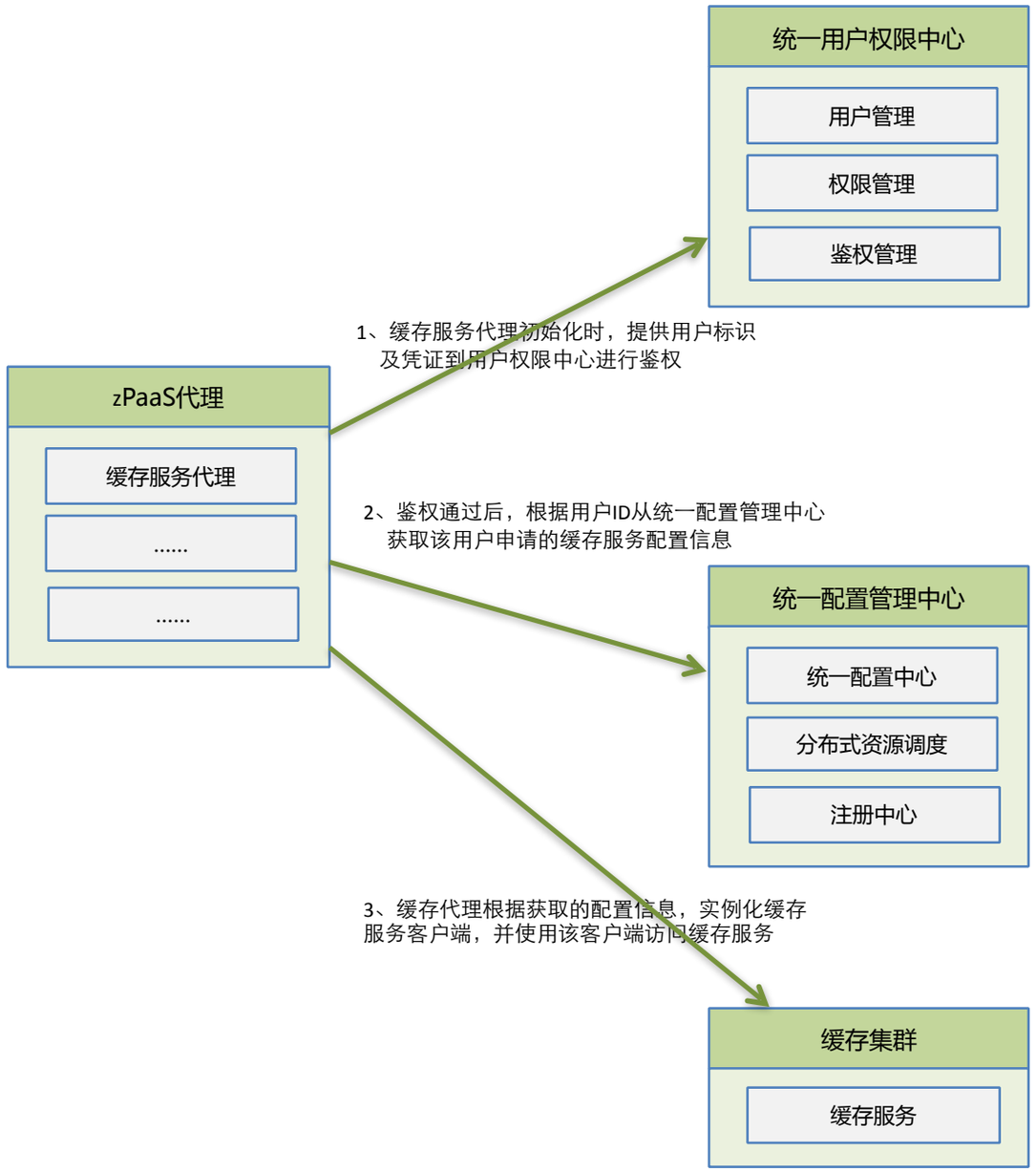
- 数据库层
    - 基于分布式数据库引擎，提供分库分表、主备及读写分离的能力
    - 配置准实时的主备同步
    - 通过修改统一配置中心相关的配置，能够实现在线故障切换
- zPaaS平台
    - 所有组件都采用集群部署模式并基于统一配置中心构建
    - 通过修改统一配置中心相关的配置，能够实现在线故障切换
    - 集群模式可以提供高可用和高并发处理能力
- 服务集群
    - 采用多实例部署模式，通过zookeeper实现负载均衡
    - 实例采用状态模式部署，并能够被zookeeper侦测，单实例的故障不影响系统可用性
- Web集群
    - 采用多部署部署，通过负载均衡集群实现负载均衡
    - 采用统一session管理方案，使每个实例都处于无状态模式，单实例的故障不影响系统可用性

关键组件介绍



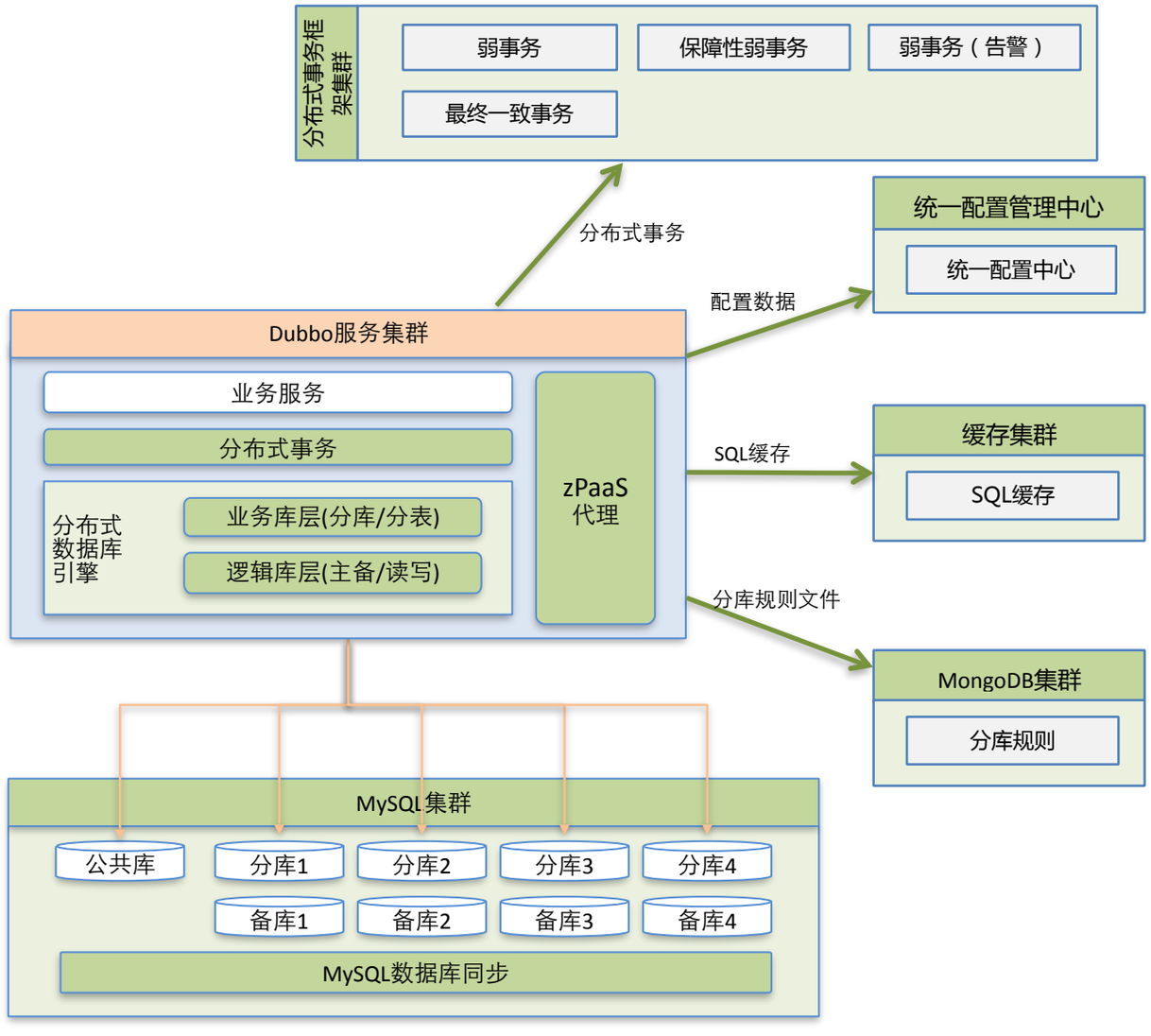
平台核心-统一配置管理中心

- 统一配置中心
  - zPaaS平台的核心组件，统一管理、分配和推送平台提供的各类服务，能够提供实时变更的能力
  - 使用Zookeeper作为zPaaS平台的统一配置管理中心
- 注册中心
  - 使用Zookeeper作为Dubbo注册中心
  - 使用Zookeeper作为Kafka注册中心
- 资源调度中心
  - 使用Zookeeper作为分布式资源的调度中心
  - 通过资源调度中心的调度，实现分布式资源的并发控制及动态调度
  - 如：可以将后台扫描程序需要描述的16张分库作为一种分布式资源，资源调度中心可以将这16个分布式资源平台分配到所有的扫描程序实例上，当有新的实例加入或退出时，资源调度中心能够自动重新进行资源调度



主要组件-分布式关系数据库服务

- 垂直水平划分方案
  - 公共库：不进行分库分表，但是提供主备及读写分离，该库只提供查询操作，部分表上载到高速缓存
  - 业务库：分成多个逻辑分库，每个分库存放多个分表，即每张表拆分为多张分表，分别放到多个分库中；每个逻辑分库都对应一个或多个可以提供只读功能的备库；
  - 主备库之间实现准实时的数据同步
- 分布式数据库引擎
  - 业务库层：实现对分库、分表的封装，如示例业务库，包含4个逻辑库，逻辑分库1、逻辑分库2、逻辑分库3、逻辑分库4
  - 逻辑库层：实现主备及读写分离，如逻辑分库1，包含物理库，分库1（主）和备库1（从）
  - 物理库对应具体的数据库实例，如物理分库1，每个物理分库中可以包含多张分表，如包含0001到0004的4张分表
- 分布式事务
  - 业务域内使用JDBC事务、弱事务、保障性弱事务以及弱事务（告警）等几种模式来处理事务
  - 业务域间使用最终一致事务来处理事务



主要组件-分布式数据库主要特性

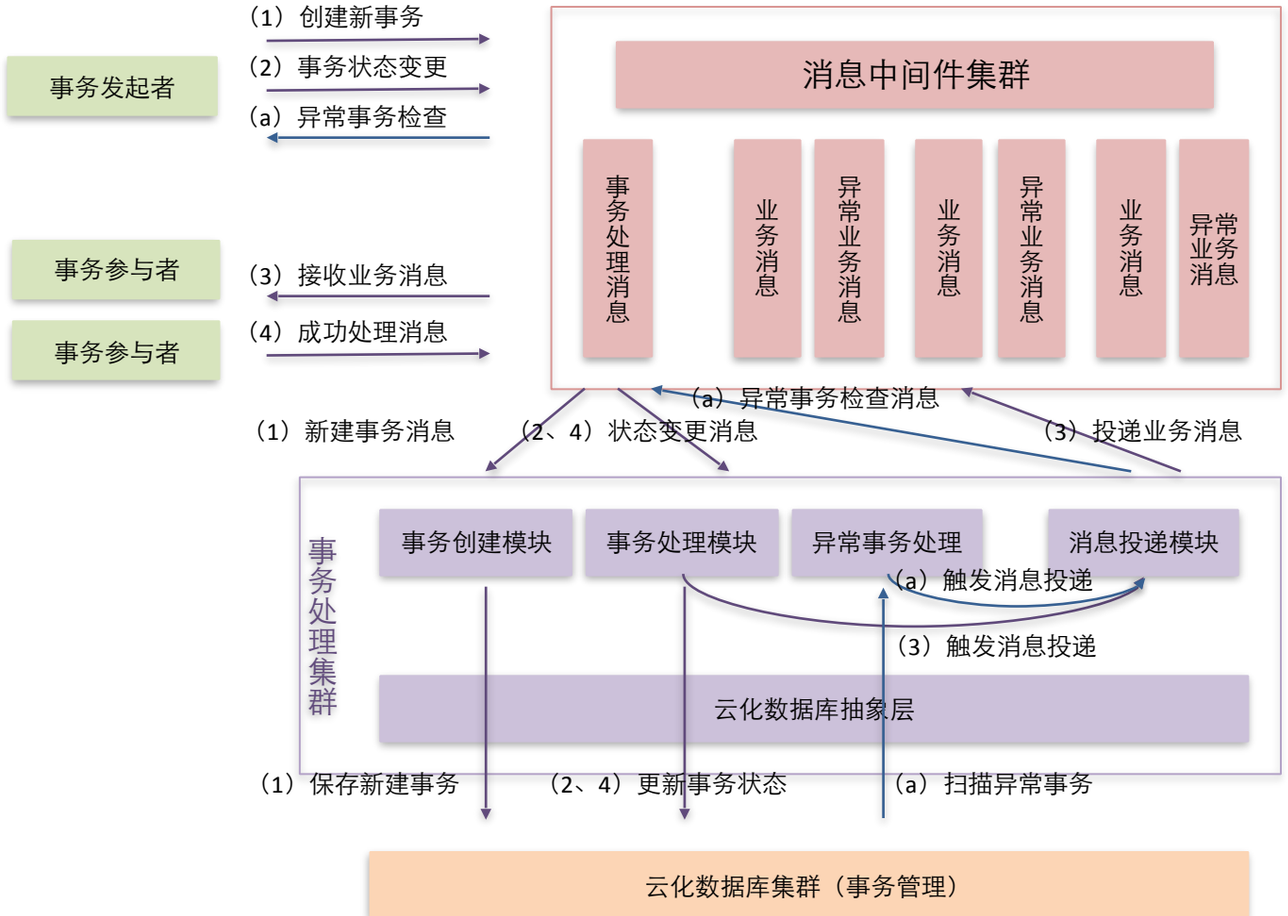
读写分离	一主（读写）多从（只读）	提供从读库读取数据的功能，并支持权重设置
分布式数据库抽象	多库操作/单库操作控制	提供是否支持多库操作的控制
	基本SQL支持	支持分布式数据库环境下的基本DML语句
	基本DDL语句支持	支持分布式数据库环境下的基本DDL语句
	集合语句的支持	支持分布式数据库环境下的count、max、min、avg等集合语句
	SQL子句的支持	group by、having、order by等
	子查询的支持	select子查询
	支持分库分表	支持分库分表
	支持读写分离	支持读写分离
	支持sql cache	支持sql cache
	在线分库规则变更	在线分库规则变更
	多租户数据库支持	在数据库层支持多租房的限制
	弱事务一致性支持	弱事务一致性支持
分布式事务框架	最终事务一致性框架	提供最终事务一致性支持
	保障性弱事务框架	提供保障性弱事务支持
数据重新分布工具	数据重新分布工具	分库分表规则发生变化后，支持数据的重新分布
监控管理工具	监控管理工具	对分库式数据库进行监控和管理
运维工具	表结构变更	
	数据库启停管理	
	数据库部署管理	
	数据库参数配置管理	
准实时主从数据同步	准实时主从数据同步	准实时主从数据同步

主要组件-分布式事务框架

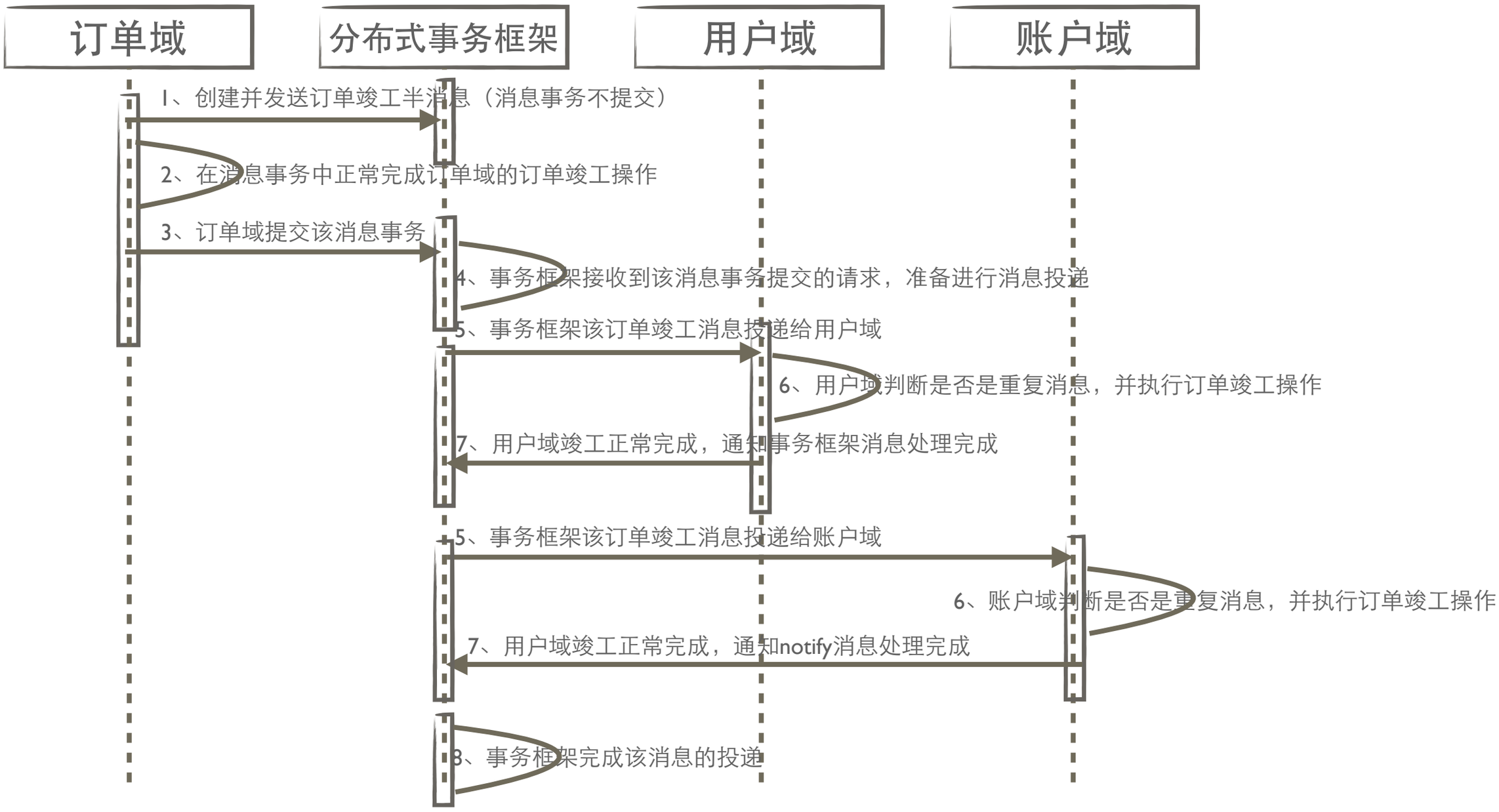
- 分布式环境下常见的事务类型
  - 强事务一致性
    - XA事务
    - 优点：能够保证事务的强一致性
    - 缺点：在分布式环境下效率比较低
  - 弱事务一致性
    - 将业务操作分成两部分，先尝试执行所有的业务操作，在所有的业务操作执行成功的情况下，在最后统一提交所有的事务，包括弱事务及两阶段提交
    - 优点：实现简单，效率高，正常情况下能够保证事务一致
    - 缺点：异常情况会会产生不致性事务，弱事务不支持分域部署的方式，两阶段提交在子事务比较多的情况下效率较低
  - 最终事务一致性
    - 在弱事务一性的基本上，引入消息系统，在异常情况下也能保证事务最终一致
    - 优点：效率高，在异常情况下也能保证事务最终一致，部署上无特殊要求
    - 缺点：实现相对复杂，一段时间内会存在事务不一致的情况

主要组件-最终一致事务

- 最终一致事务
  - 最终一致事务将整体事务拆分成主事务与几个独立的子事务，只有在主事务成功的情况下，才通过消息系统将事务的业务消息投递给子事务，从而触发子事务的执行
  - 当子事务执行发生异常时，最终一致事务框架将重得将事务消息投递给子事务，直到子事务成功执行，最终达到事务一致状态
  - 最终一致事务将原来整体的事务异步化处理后，子事务之间存在一定的延时，适用于业务子系统之间的事务整合

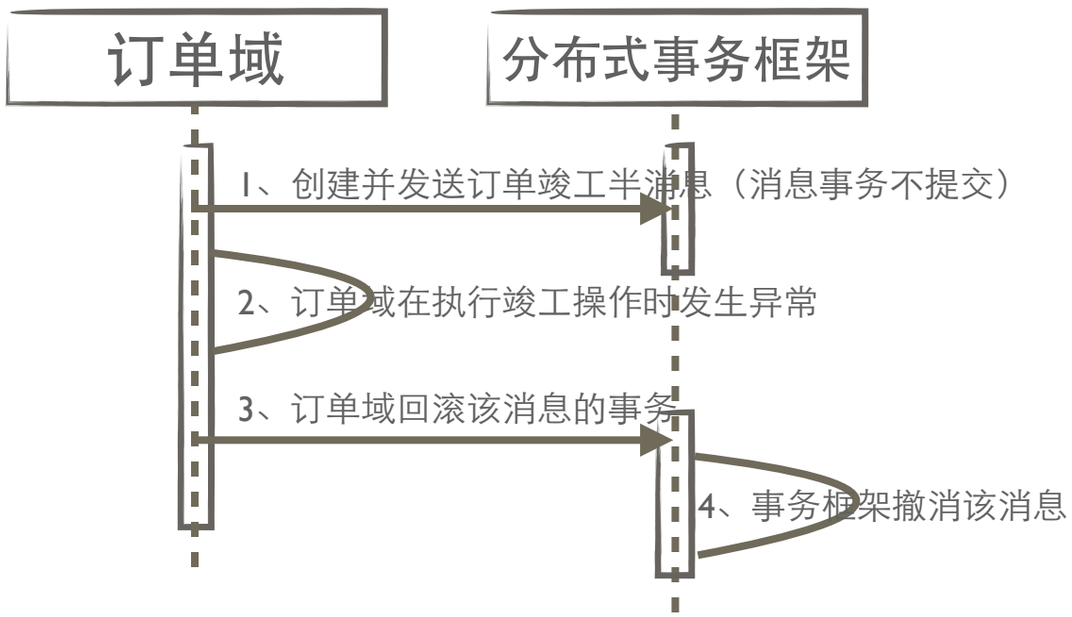


主要组件-最终一致事务

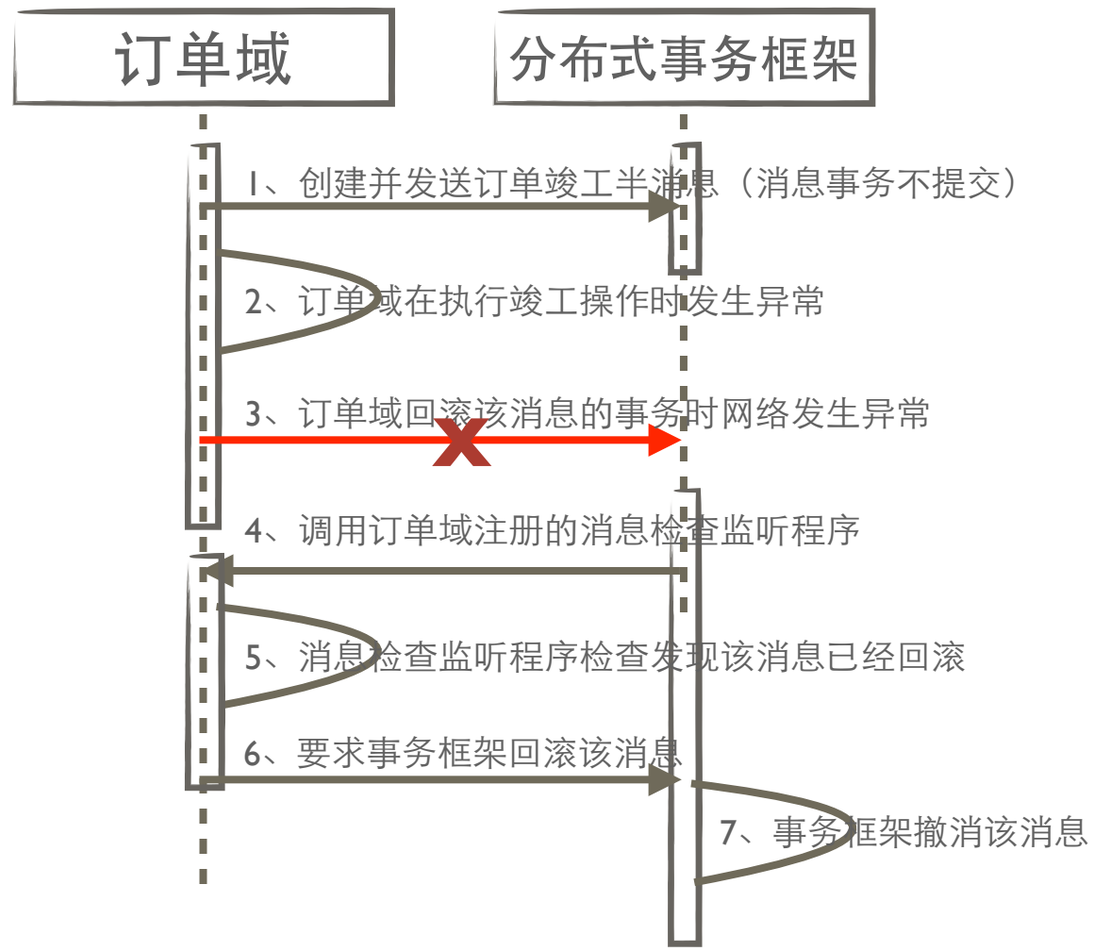

订单域
分布式事务框架
用户域
账户域
1、创建并发送订单竣工半消息（消息事务不提交）
2、在消息事务中正常完成订单域的订单竣工操作
3、订单域提交该消息事务
4、事务框架接收到该消息事务提交的请求，准备进行消息投递
5、事务框架该订单竣工消息投递给用户域
6、用户域判断是否是重复消息，并执行订单竣工操作
7、用户域竣工正常完成，通知事务框架消息处理完成
5、事务框架该订单竣工消息投递给账户域
6、账户域判断是否是重复消息，并执行订单竣工操作
7、用户域竣工正常完成，通知notify消息处理完成
8、事务框架完成该消息的投递

主要组件-最终一致事务-发起端异常流程

业务异常回滚消息



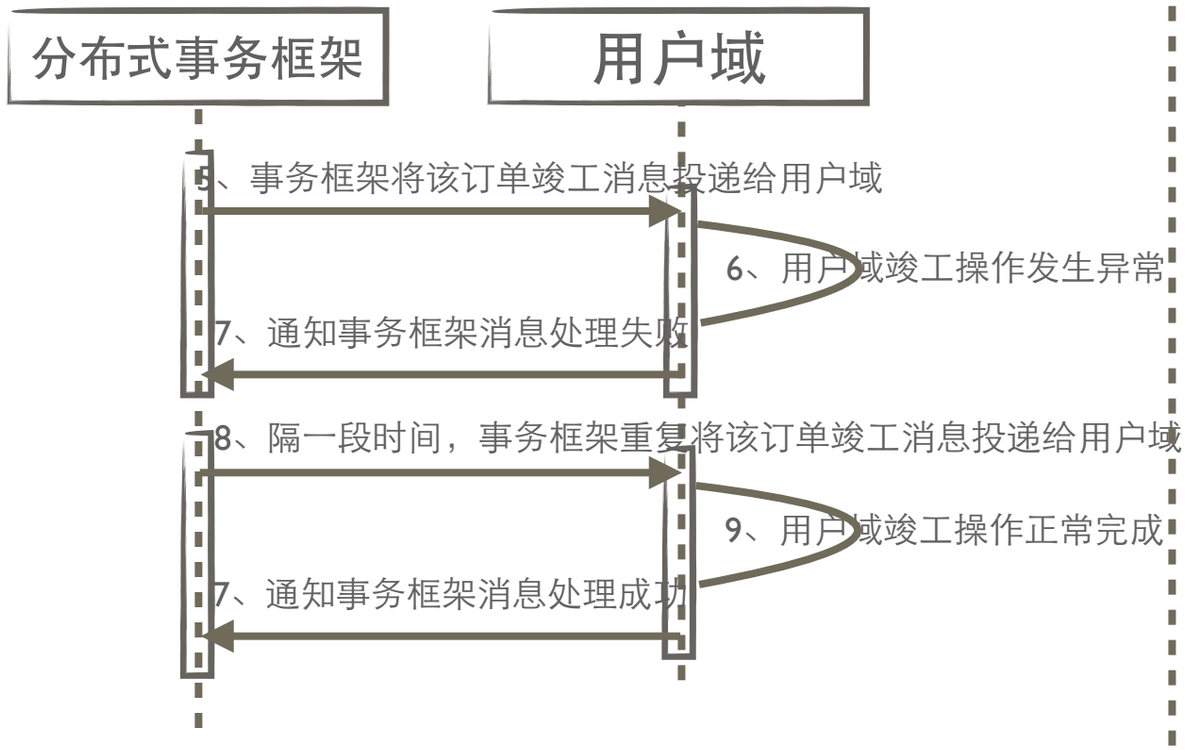
系统异常回滚消息



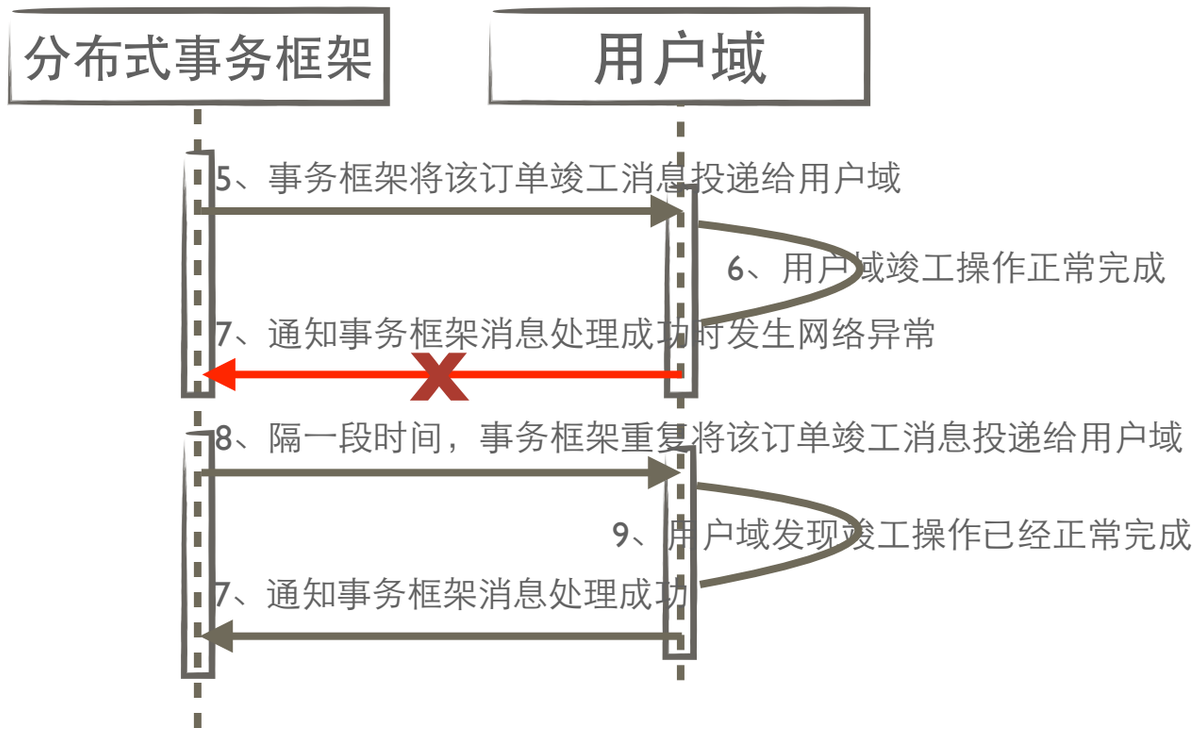
注：系统异常造成第3步消息事务提交失败的异常流程与该流程相同

主要组件-最终一致事务-投递端异常流程

业务异常造成消息重复投递



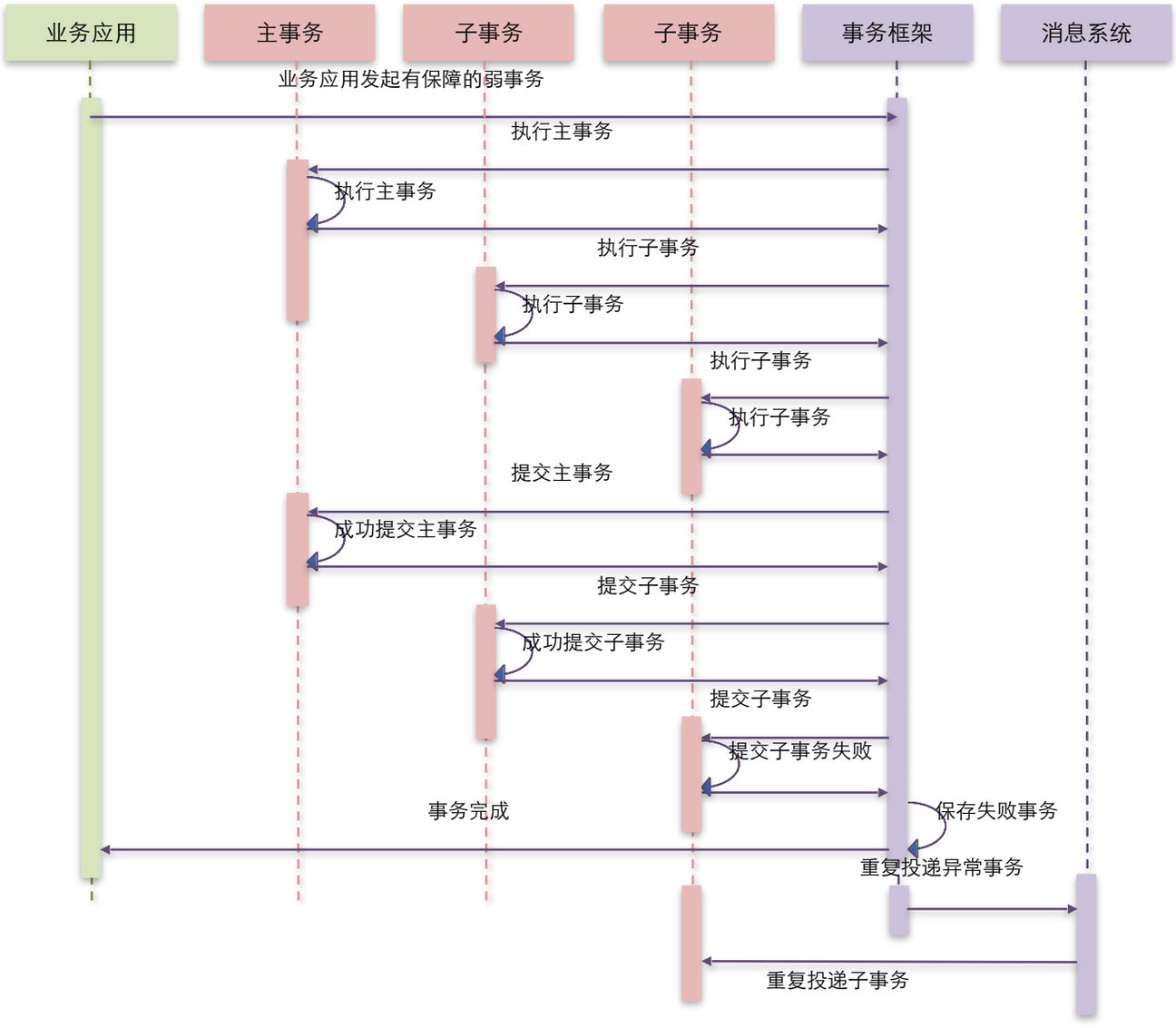
系统异常造成消息重复投递



注1：Notify通过消息的重复投递来保证事务的最终一致

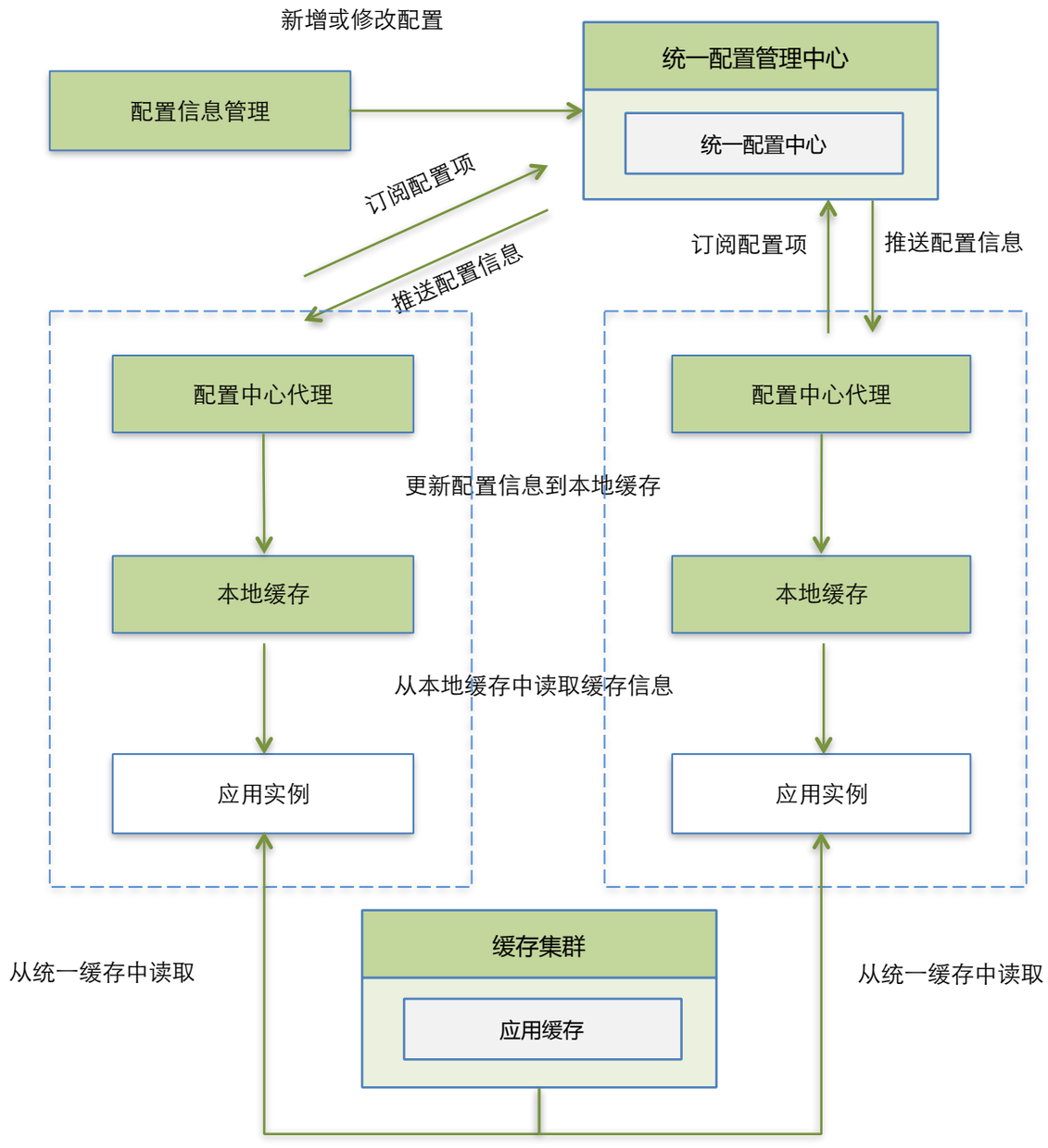
主要组件-保障性弱事务

- 保障性弱事务
  - 保障性弱事务是在弱事务的基础上结合最终一致事务的消息保障机制，使分布式事务即有弱事务的同步和高效的特点，又在事务发生异常时能够像最终一致事务那样得到最终一致的保障
  - 保障性弱事务也将整体事务拆分成主事务和几个子事务，尝试执行所有的主事务和子事务，但不提交；在所有事务的尝试都成功的前提下，首先提交主事务，如果主事务提交成功，则开始逐个提交所有的子事务。
  - 当有子事务的事务提交失败时，则事务框架将该失败事务的业务消息保存到事务数据库中，再重复将该事务的业务消息投递给失败的子事务，直到子事务成功为止，从而最终使事务达到一致
  - 保障性弱事务适合于业务子系统内部的分布式事务控制



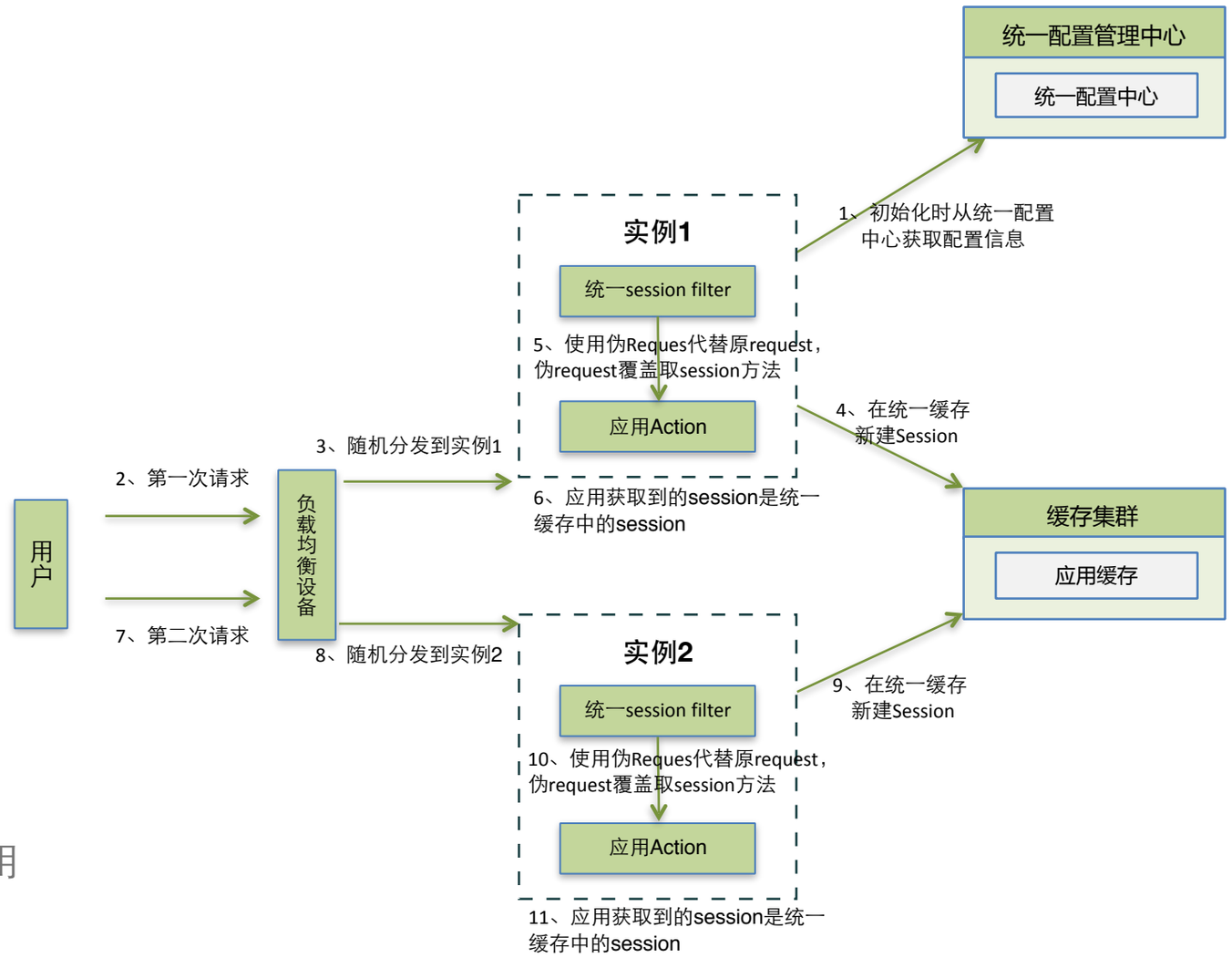
主要组件-缓存服务

- 应用缓存采用本地缓存和统一缓存结合的方案
- 本地缓存
  - 主要用于存储相对静态的配置性数据
  - 通过统一配置中心能够实现一点变更实时同步到各应用实例
  - 通过本地缓存的方式来存储比较静态的配置性数据可以极大的减少网络访问的频次，提高系统性能
- 统一缓存
  - 主要用于存储配置性数据之外的其他静态数据和需要缓存的业务数据
  - 采用Redis集群



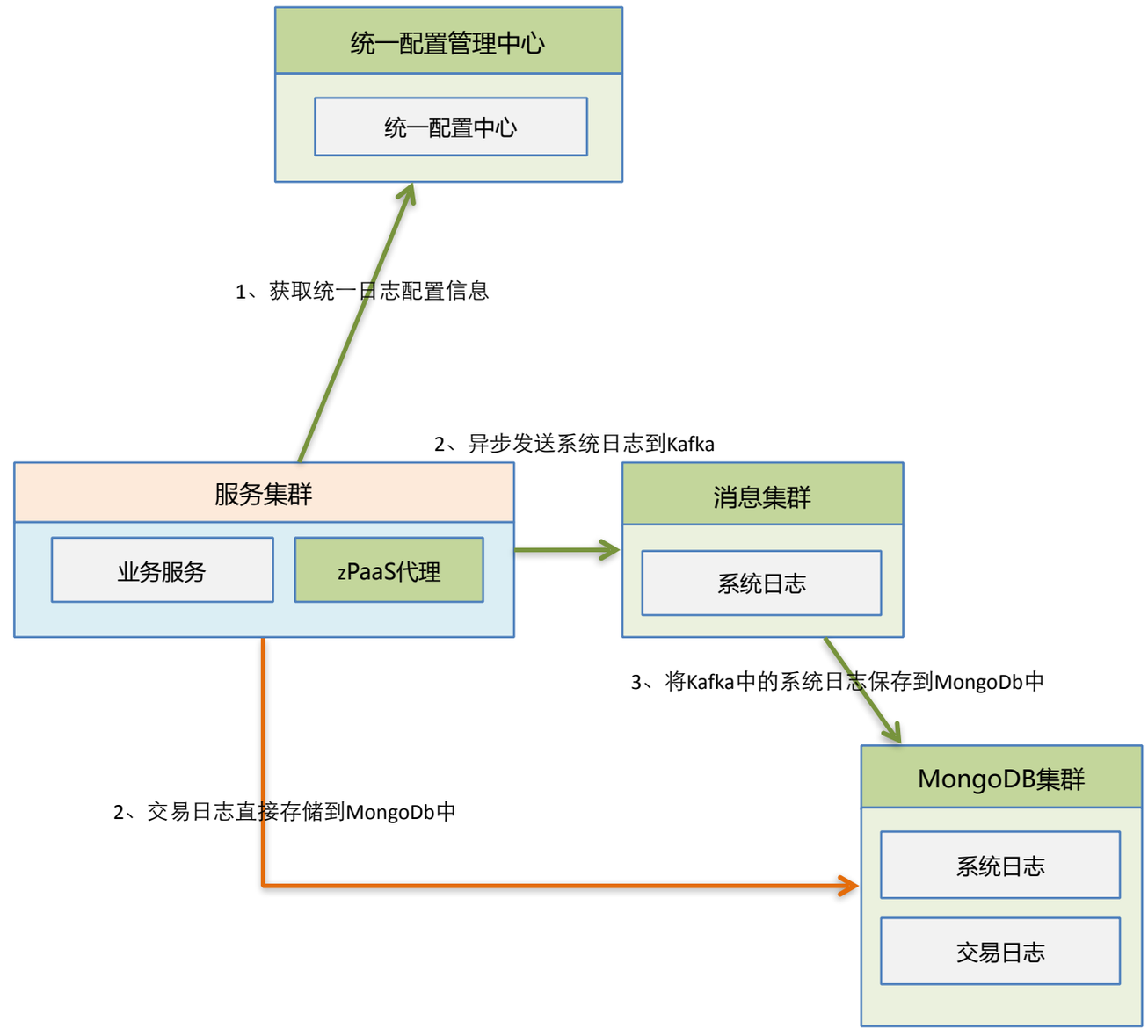
主要组件-统一Session服务

- 传统Session管理模式
  - 在web中间件服务器的JVM中保存
  - 在分布式环境下需要负载均衡设备保持会话，增加负载均衡设备的负担
  - 当某个实例宕机时，该实例上保存的Session信息将丢失
- Session复制模式
  - 通过web中间件提供的session复制功能将session信息在所有实例间同步
  - 当实例比较多时，会对网络造成很大的压力
- 统一session服务方案
  - Session保存到统一缓存中，使用filter对http session进行拦截，对应用开发无侵入
  - 负载均衡设备不需要进行会话保持，可以采用最高效的随机分发模式
  - 某个实例的宕机不会影响用户的session



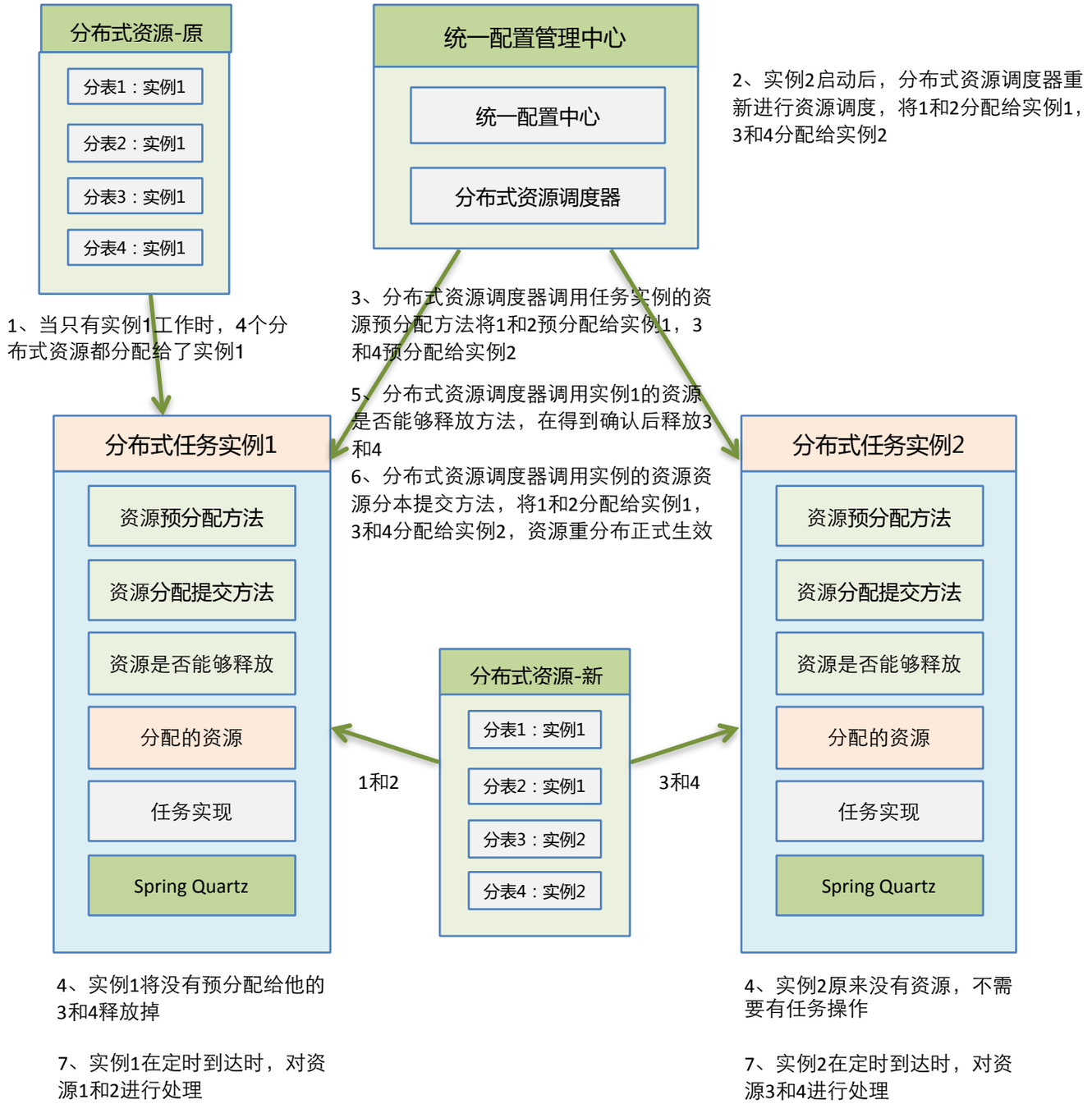
主要组件-统一日志服务

- 常规日志管理模式在分布式环境下的困局
  - 在分布式环境下，常规的日志管理方式所有的日志都分散在各个服务器上，查找问题时很难确定落到哪个服务器上
  - 用户一次交易的日志分散在不同服务器的不同文件中，无法串联用户交易上下文进行问题排查
- 统一日志管理
  - 采用MongoDB集群作为统一的日志存储库
  - 统一管理的日志包括系统日志和交易日志
  - 能够提供统一的查询界面，一点查看完整的日志
  - 系统日志
    - 使用log4j的Appender实现，对应用程序无侵入
    - 结合Kafka集群实现异步化日志记录，减轻日志记录对性能的影响
    - 使用统一的thread id串联web端与服务端的日志，实现执行堆栈重现
  - 交易日志
    - 所有原始交易报文都记录到交易日志库中，以备发生纠纷时查询



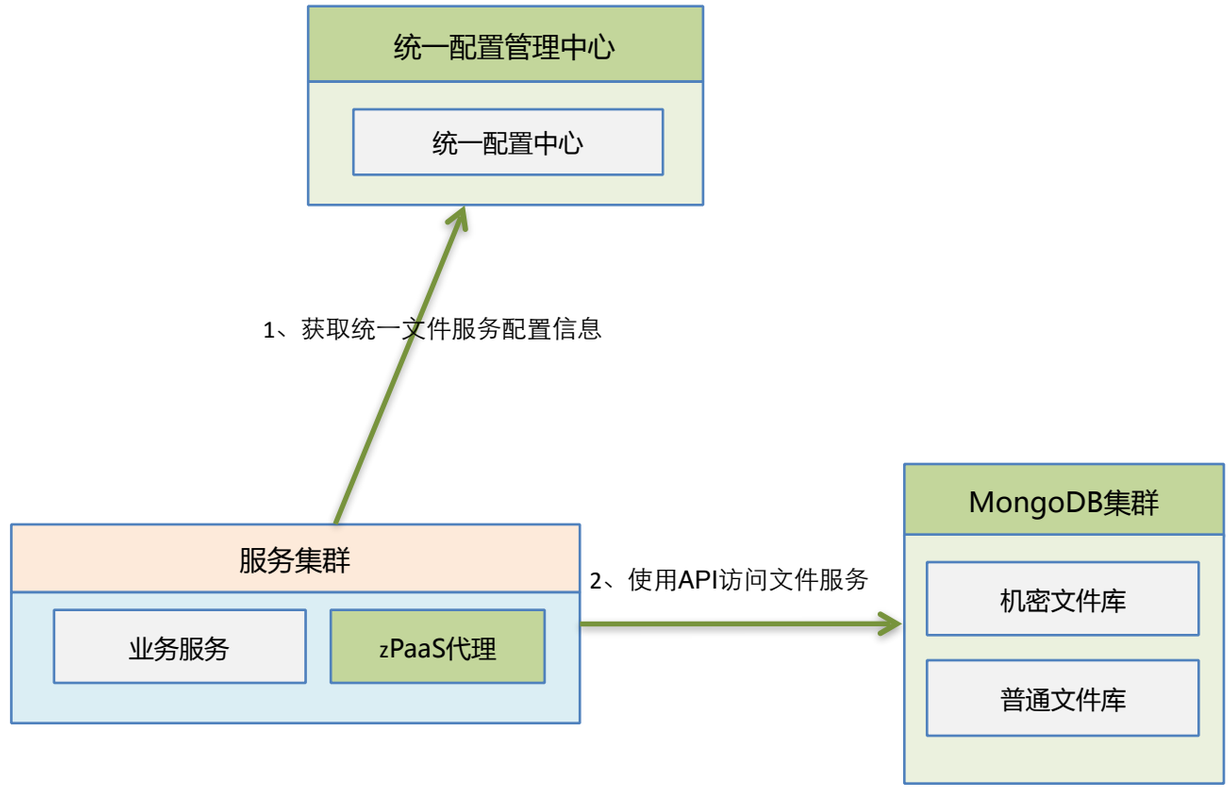
主要组件-分布式定时任务

- 定时任务框架在分布式环境下的难题
  - 并发情况下的资源竞争问题
  - 任务实例故障会影响任务的执行
  - 任务实例的弹性伸缩问题
- 分布式定时任务
  - 按资源进行并发：有几个资源，最多有几个有效任务实例，如按分表进行并发，假设有4个分表，那就有4个资源，当有一个任务实例时，4个资源都由该实例处理；当有2个实例时，每个实例处理2个资源；当有4个实例时，每个实例处理1个资源；当超过4个实例时，只有前4个实例每个处理1个资源，剩余实例处于空闲等待状态
  - 实时由分布式资源调度器进行资源调度：当有任务实例增减时，分布式资源高度器会实时进行资源的重新分配，分布式定时任务需要实现调度器要求的接口方法
  - 分布式定时任务只对分配到的资源进行处理
  - 采用Spring集成的Quartz定时任务框架



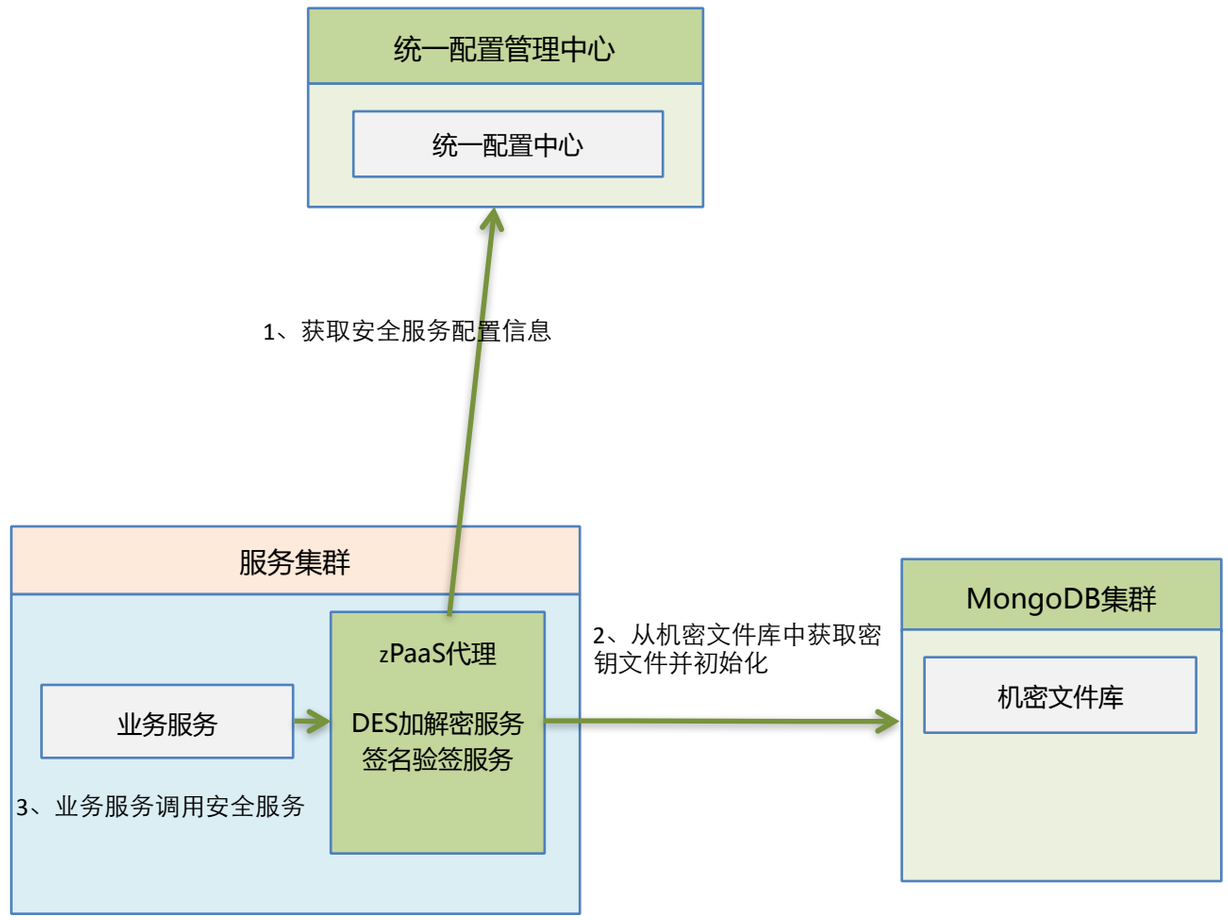
主要组件-统一文件服务

- 分布式环境下传统文件存储方式的困局
  - 本地文件存储方式
    - 存储分散、无法共享
    - 备份困难、无法统一管理
    - 没有权限控制，不能存储敏感文件
  - FTP/SFTP
    - 单服务器存储、存在性能和存储的瓶颈
    - 需要人工备份
- 统一文件服务
  - 采用MongoDB集群并采用GRIDFS作为统一的文件存储库
  - 通过MongoDB的权限控制机制，能够提供安全的文件访问权限控制
  - MongoDB集群的主备、复制组以及分片模式能够提供不同级别的系统需求，提供高可用、高性能、弹性扩展以及灾备和快速故障恢复的能力
  - 提供不同安全级别的文件存储服务



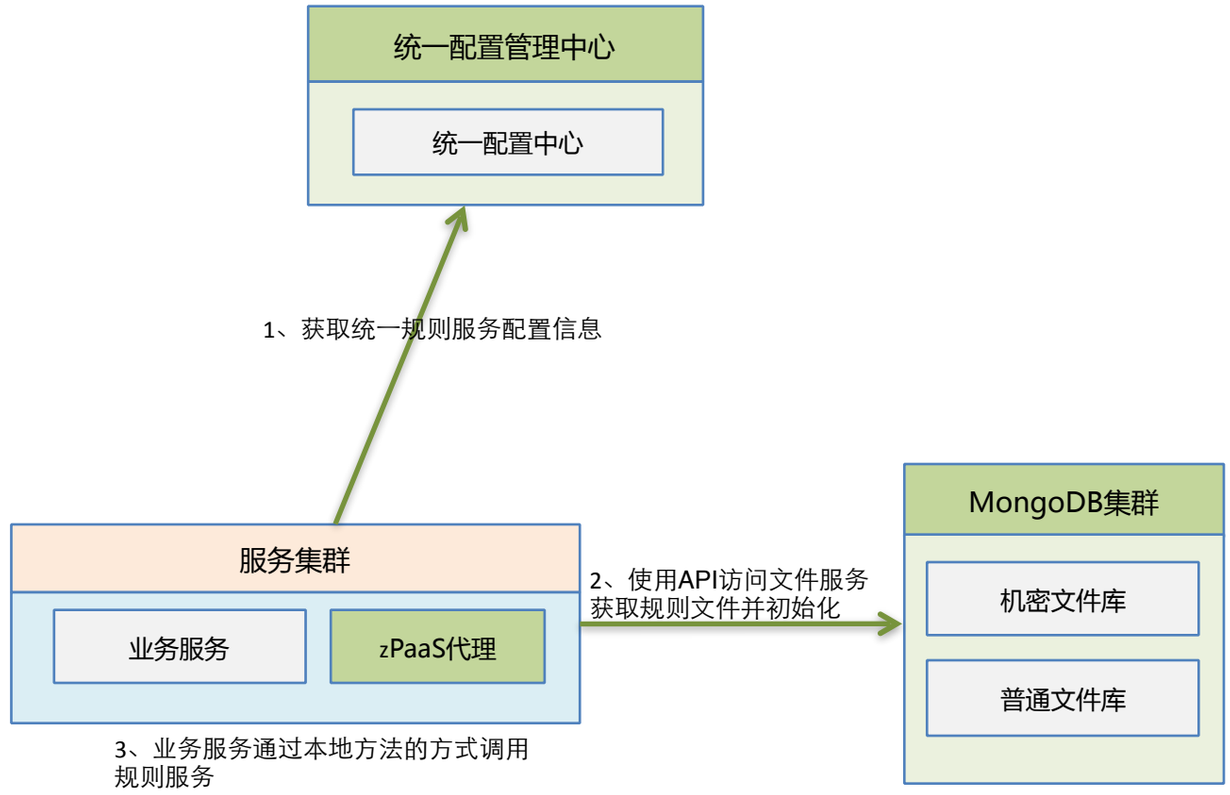
主要组件-安全类服务

- DES加密及解密服务
  - 提供多种标准的DES加解密服务
    - DES/ECB/PKCS5Padding
    - DES/CBC/PKCS5Padding
    - DES/CBC/NoPadding
  - 密钥文件在机密文件库中统一保存，不在本地存储上留存，防止密钥文件泄漏
- 签名及验签服务
  - 提供多种签名及验签服务及管理方式
    - 标准的签名及验签服务：SHA1WithRSASignatureSVCImpl
    - 全民付的签名及验签服务：UmsPaySignatureSVCImpl
    - 签名工具管理器：SignatureSVCManager
  - 公私钥文件在机密文件库中存储，不在本地存储上留存，防止私钥文件的泄漏

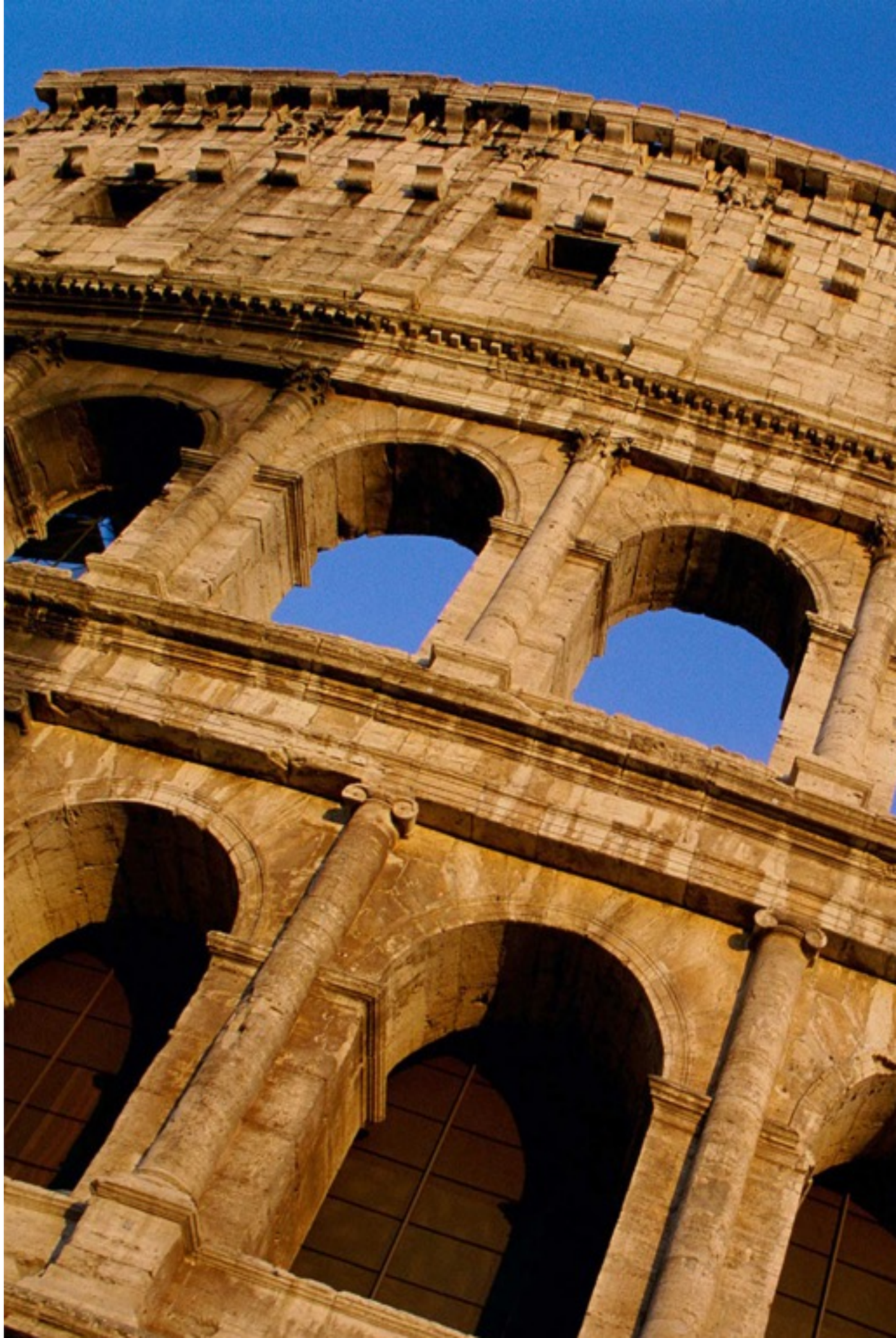


主要组件-规则服务

- 分布式环境下规则服务的常用实现方式
  - 独立服务部署形式
    - 规则服务部署在独立的服务器上，应用通过远程服务调用的方式访问服务
    - 每次规则服务的访问都涉及到网络的访问，性能相对较低
    - 规则服务与应用的部署关系比较复杂，每个规则里需要调用的方法和服务都涉及到网络的访问
  - 本地规则文件部署形式
    - 规则文件直接打在部署包中，通过本地方法调用的形式访问规则服务
    - 该方式解决了网络访问和服务调用的问题，但是带来了规则文件无法实时更新的问题
- 统一规则服务
  - 采用在MongoDB集群中存储规则文件，并在统一配置中心配置规则服务的方式，
  - 规则服务在应用本地，当规则服务初始化时，根据统一配置中心配置的规则服务信息从文件库中加载规则文件
  - 应用通过本地方法的方式调用规则服务
  - 通过统一配置中心以及文件库的配合又能达到在线变更规则文件的目的



开发注意事项



分布式数据库设计原则

- 垂直切分：即业务域的划分
    - 划分原则：域内的实体相对内聚、域间的实体相对松耦合。基本上可采用按系统业务域的方式来划分。
    - 以CRM系统来举例，CRM系统可划分为：客户域、订单域、用户域、账户域、产品域、公共域等。
- 水平切分：即对业务域内的数据再进行横向切分，进行分库分表
    - 水平切分的参考指标：
        - 预计或规划的单表数据量可能会超过500万条
        - 预计或规划的单库访问量Q/TPS可能会大于5000（机械硬盘）或大于3万（SSD）
    - 以CRM系统为例，客户域、订单域、用户域、账户域适合进行水平切分，产品域和公共域不切分，但数据加载到缓存中
    - 切分维度选择原则：
        - 核心字段原则：业务域内的核心关联字段，如客户域的客户ID、用户域的用户ID
        - 频繁使用原则：结合业务情况，使用比较频繁的查询字段，如资源域的省分（或联合地市，选号场景）
        - 数据平均原则：采用的分库字段需要能够将数据相对比较均匀的打散到各个分表中
        - 次要维度根据业务实际情况具体分析
        - 索引表：以索引字段作为分库键
    - 索引表使用原则：在没有分库键的情况下，需要用来查询的重要字段，需要按该字段建立索引表，维护主表的时候需要同时更新索引表
        - 以常用的客户查询为例，经常需要用证件号码对客户进行查询，此时需要建立证件号码与客户ID的索引表，该索引表以证件号码为分库键，查询时，先根据证件号码从索引表查询获得对应的客户ID，再根据客户ID查询详细信息
- 物理分库主从同步：使用准实时数据同步工具进行主从备份、对于只读事务可以使用读写分离的方式分担主库的压力

分布式数据库使用注意事项

- 使用简单语句
    - 简单语句，单表操作
    - 原则上不允许使用关联查询，如果一定要使用关联查询，则关联表数不要超过3张并且涉及的数据都要落到同一维度的分表，否则引擎不保证查询到正确的数据
    - 所有语句都需要带上分表字段并能唯一确定一张分表，原则上不允许多分表操作
- 使用索引表
    - 当无法确定分库键时，需要先使用索引字段从索引表中查询到分库键
    - 如，先使用证件号码查询客户证件号码索引表查询得到客户ID，再使用客户ID从客户主表中查询客户详细信息
- 序列
    - 需要使用分布式数据库引擎提供的序列服务，不能直接使用数据库提供的sequence
    - 只保证唯一，不保证连续且存在序列值浪费的情况
- 分页
    - 有限支持分页功能，查询的数据必须落到同一维度的分表且必须带有分库字段
    - 实践建议：应用自己可以实现分页功能，假设有16张分表，假设每页显示10条，那第可以从每个分表获取10条记录，总共160条，应用再在内存中进行分页处理，当显示11页时，再一次性取出160条。

分布式数据库使用注意事项续

- 查询功能弱化
  - 能确定分库键时，正常查询
  - 不能确定分库键但能确定索引字段的情况下，先查索引表再查主表
  - 不能确定分库键也不能确定索引字段的情况下，要么会造成全分表查询（效率低），要么放弃该维度的查询功能
- 不支持批量SQL
- 不能使用存储过程和触发器
- 报表方案：需要制定专门的方案
- 开发时不需要关注落到哪个分库和哪张分表，但需要关注分库字段及索引字段

分布式事务框架使用注意事项

- 适用场景：涉及多个JDBC原子事务的分布式事务场景。最终一致事务适合业务域间（多个JVM间）的事务控制；保障性弱事务适合业务域内（一个JVM内）的事务控制。
- 分布式事务拆分原则：最终一致事务和保障性弱事务都将整个分布式事务拆分成1主多从的几个子事务，每个子事务都是一个原子的JDBC事务
- 事务异常保障机制：
    - 主事务是否提交检查：在某种异常场景下，分布式事务框架不确认主事务是否已经提交，因此在主事务则需要实现主事务是否提交的检查方法
    - 子事务重复投递处理：在某种异常场景下，分布式事务框架无法确定子事务是否已经正常提交，框架可能会重复投递子事务消息，因此在子事务侧需要能够正确处理重复投递的子事务消息。可以通过记录的ID、时间或版本号进行判断。
- 无序投递：分布式事务框架投递的事务消息是无序的，因此不能依赖分布式事务框架进行顺序相关的业务控制
- 一定时间内的事务不一致：
    - 最终一致事务是异步事务，在短时间内可能存在事务不一致的情况；
    - 当发生异常时，最终一致事务和保障性弱事务，在一段时间内都可能存在事务不一致的情况
- 事务的参与方，包括主事务和子事务，其group_id要唯一

谢谢！