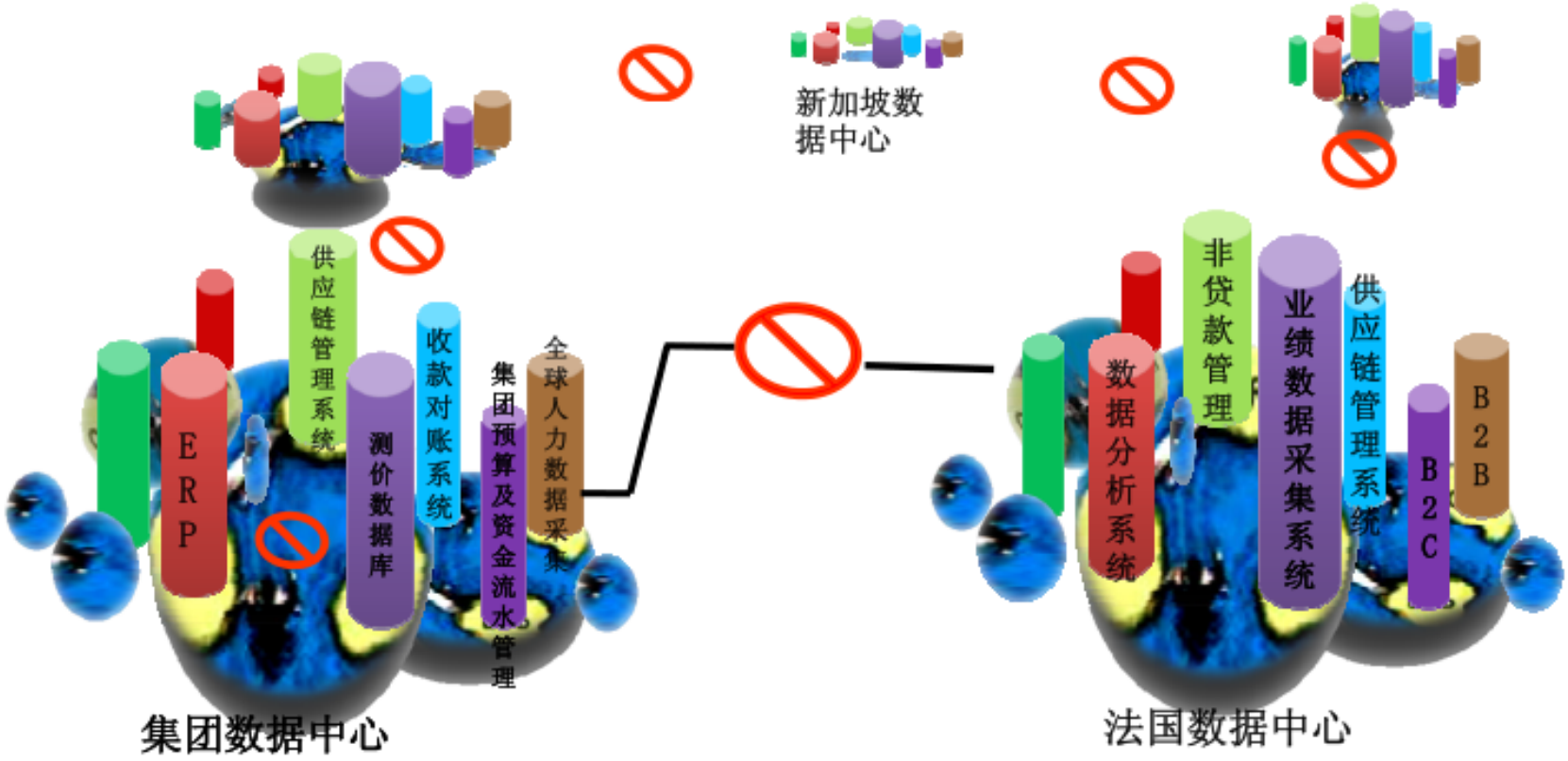
基于ActiveMQ的消息总线逻辑与物理架构设计详解

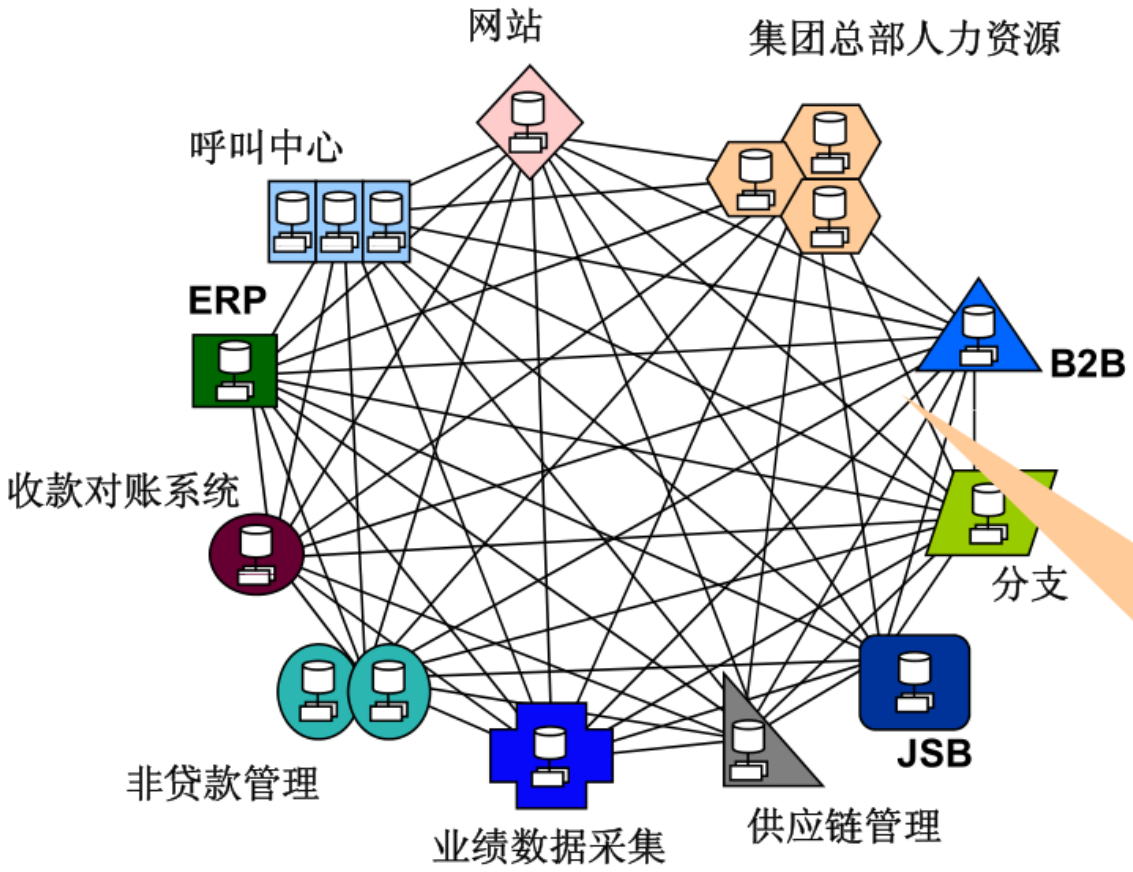
王金剑

问题与现状

往往为了连接这些信息孤岛，又产生了新的信息孤岛



问题与现状



- 系统之间直连，极端情况下n*(n-1)/1条
- 服务的变动会影响该服务的所有使用者

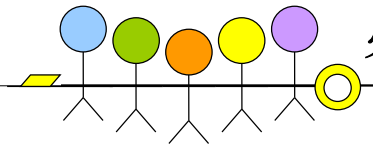
* 系统之间耦合紧密，牵一发而动全身，发展下去难以维护，不敢维护，无法维护

问题与现状

综上所述，根据我们的了解，天狮集团信息化瓶颈主要表现在以下方面：

	根本原因
全局数据难于获取	1 多套数据，数据分散，难于采集
数出多门	2 数据标准不统一
信息交换困难、复杂	3 蜘蛛网式集成，应用集成标准，集成方式不统一
数据不完整、不准确	4 数据分析缺乏体系
关键KPI指标无法及时掌握	5 需要领导和业务导向
数据分布分散，难以管理；历史数据占用空间过多	6 缺乏数据生命周期管理
数据存在泄漏风险，影响企业风控管理	7 缺乏数据审计监控手段与机制

解决方案



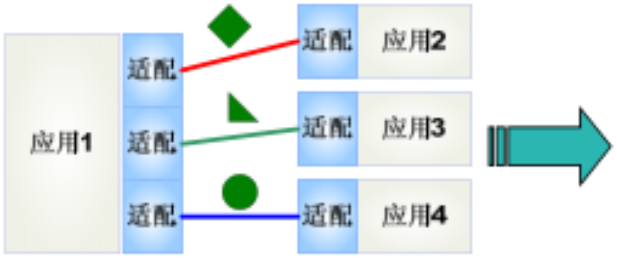
火龙果讲堂

uml.org.cn

针对上述问题,设计的解决方案包括建立总线,标准,架构等。

全局数据难于获取	1 多套数据，数据分散，难于采集和应用	建立数据服务总线
数出多门	2 数据标准不统一	建立数据标准
信息交换困难、复杂	3 蜘蛛网式集成，应用集成标准，集成方式不统一	建立企业服务总线
数据不完整、不准确	4 数据分析缺乏体系	建立数据体系，服务企业应用
关键KPI指标无法及时掌握	5 需要以领导和业务导向	数据规划，业务导向数据模型，数据架构
数据分布分散，难以管理；历史数据占用空间过多；应用数据各自为政	6 缺乏数据生命周期管理	建立数据治理手段

解决方案

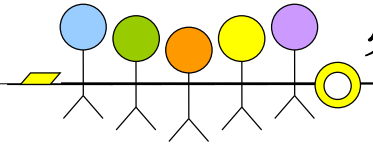


未利用平台时的应用"紧耦合集成"

利用平台时的应用"松耦合集成"

- 系统之间通过服务总线连接，接口数从n*(n-1)变成了n
- 服务的变动仅会影响该服务的适配器

消息中间件基本介绍

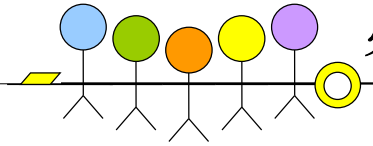


消息中间件技术有两个核心功能：

- 异步
  - ✓ 增强高并发处理能力
  - ✓ 提高工作效率
  - ✓ 提升用户体验
- 解耦
  - ✓ 增强系统可用性和可靠性
  - ✓ 增强系统可扩展性

第一问题：为什么要用消息中间件？

消息中间件基本介绍



亚马逊公司SOA软件设计开发原则

1、所有团队，要把他们的数据和软件功能通过服务接口对外公开；
2、团队之间的沟通只能通过这些接口；
3、不允许通过任何别的方式通讯：不能直接连接，走后门，不能直接读别的团队的数据，不能共享内存；
4、服务接口的后端的软件技术的选择，没有关系；
5、所有的服务接口设计时，都必须具备一个能力，允许日后让外界第三方开发者调用。没有任何例外；
6、不听话的人，将被开掉。

曾经在亚马逊工作了六年，2005年跳槽到谷歌的软件工程师，Steve Yegge 在2011年的一封公开信内介绍：

亚马逊的招聘流程本质上有缺陷。每个团队自己雇人，所以不同团队的员工招聘水准差别很大……，亚马逊的办公场所，污垢遍布，不花一分钱用于装饰……，亚马逊的程序代码一团糟，没有任何工程标准，完全看相关团队的选择……，也许有两百个不同的角度方式比较这两个公司，但除了三个以外，谷歌在所有方面都完胜亚马逊。

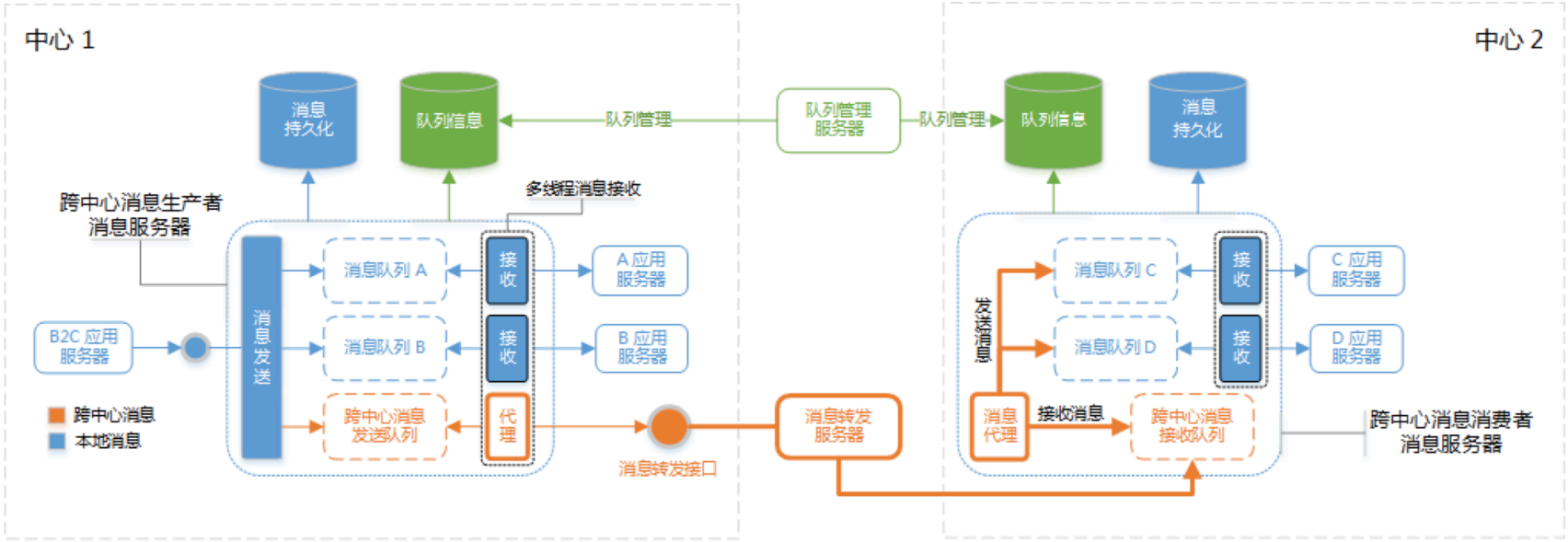
但是亚马逊有一件事，做得非常非常好，这远远弥补了亚马逊所有的政治上的，哲学上的和技术上的失误，2002年前后，贝索斯给所有员工发布了一个命令，关于内部软件设计的命令。

消息中间件基本介绍

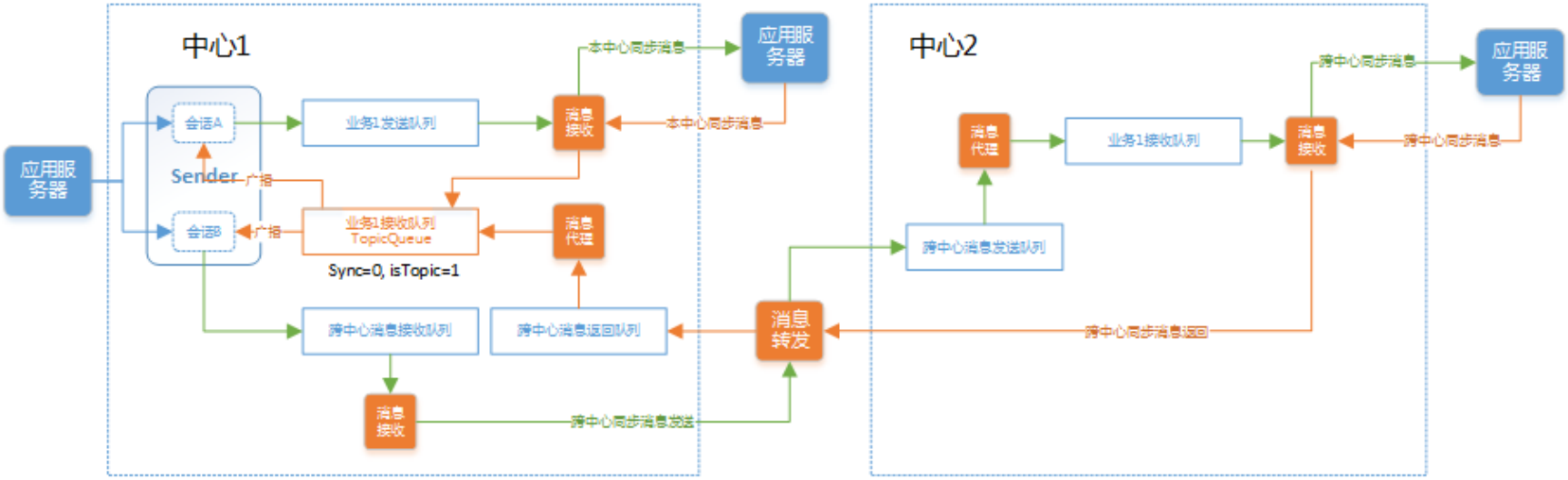
使用SunlightMQ平台可以实现企业应用系统内部解耦和服务共享，提高团队敏捷度和开发效率，逐步使企业信息中心整体实现“软件即服务(SOA)”的设计思想。
软件即服务(SOA)的本质：是一个运用于企业内部一组应用而非单个应用的系统软件架构；重点关注网络和系统间的接口，而非实现；进行一切必要的标准化，确保流畅的交互和偶尔的重用；别无其他。

应用系统前端全部采用静态页面（HTML+JavaScript），使用HTTP+JSON作为与后台通讯的协议，SunlightMQ负责交易请求的转发。交易平台可以使用多种语言开发（C#/java/PHP/...）。应用系统前端部署时，不同模块的静态页面和JS分开部署，同时也可以通过统一门户将来自不同服务器的页面资源整合在一起。

SunlightMQ 组件架构图

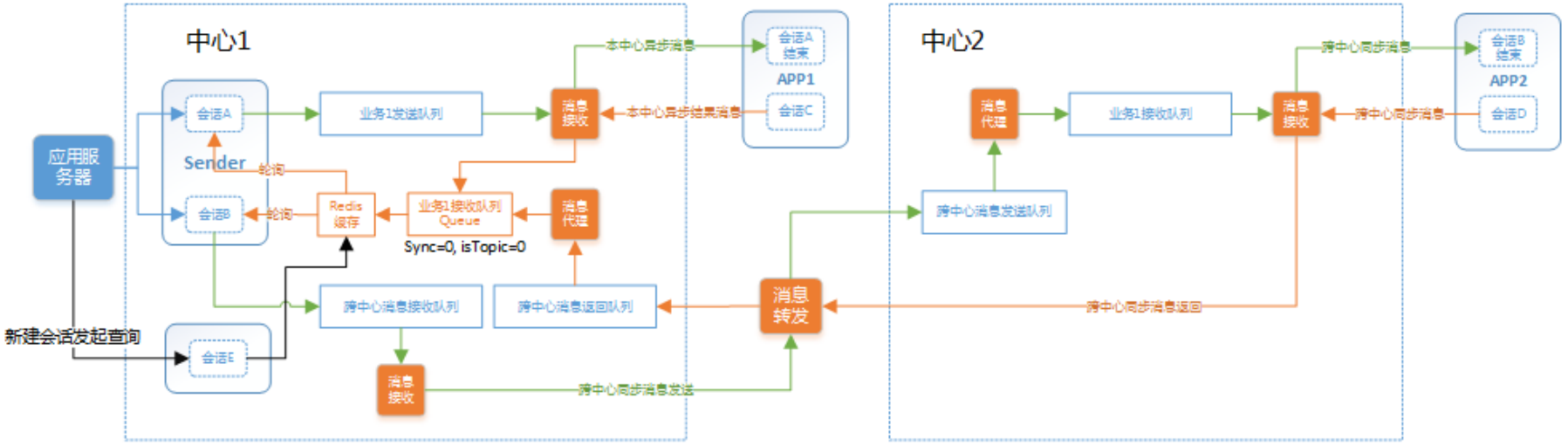


消息流转架构图-同步消息



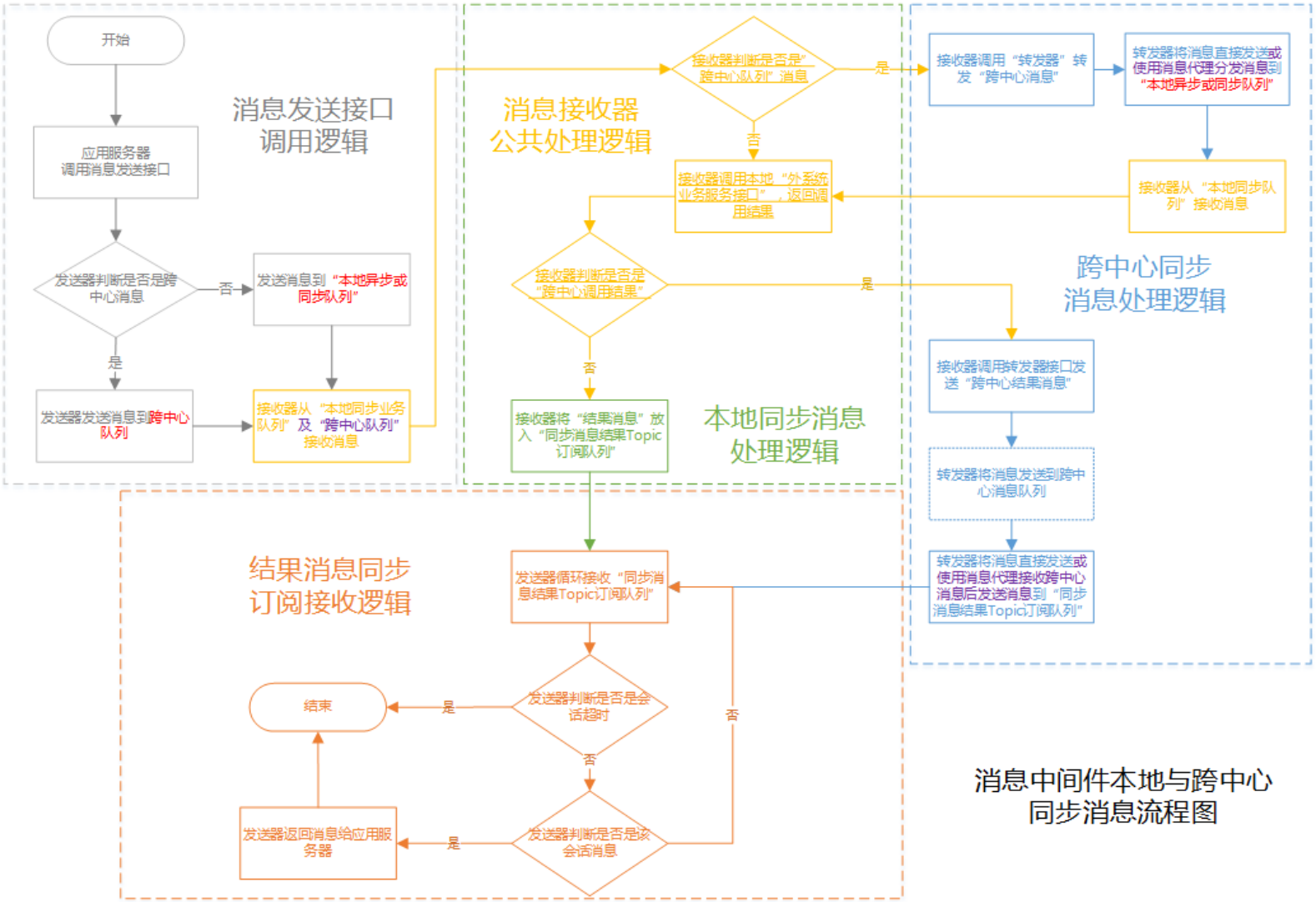
同步消息返回使用Topic类型队列，会话Receive 到消息后根据 SessionID 判断是否是该会话的返回消息

消息流转架构图-异步消息



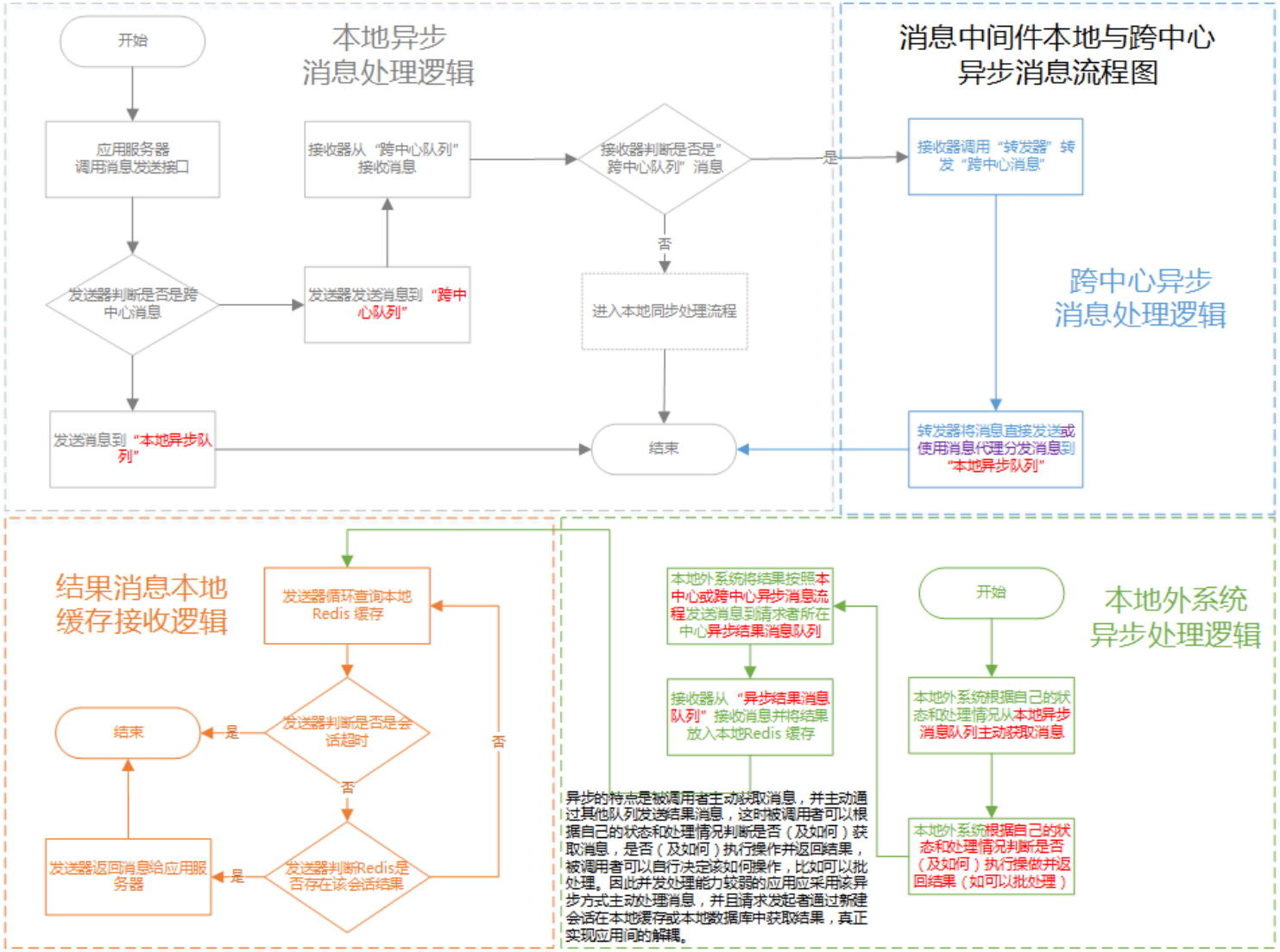
异步消息返回使用普通队列，被调用者新建会话按照异步消息发送流程，主动发送结果消息到Redis缓存，会话轮询Redis缓存，根据会话唯一标识符判断是否存在该会话的返回结果。

消息流程图-同步消息

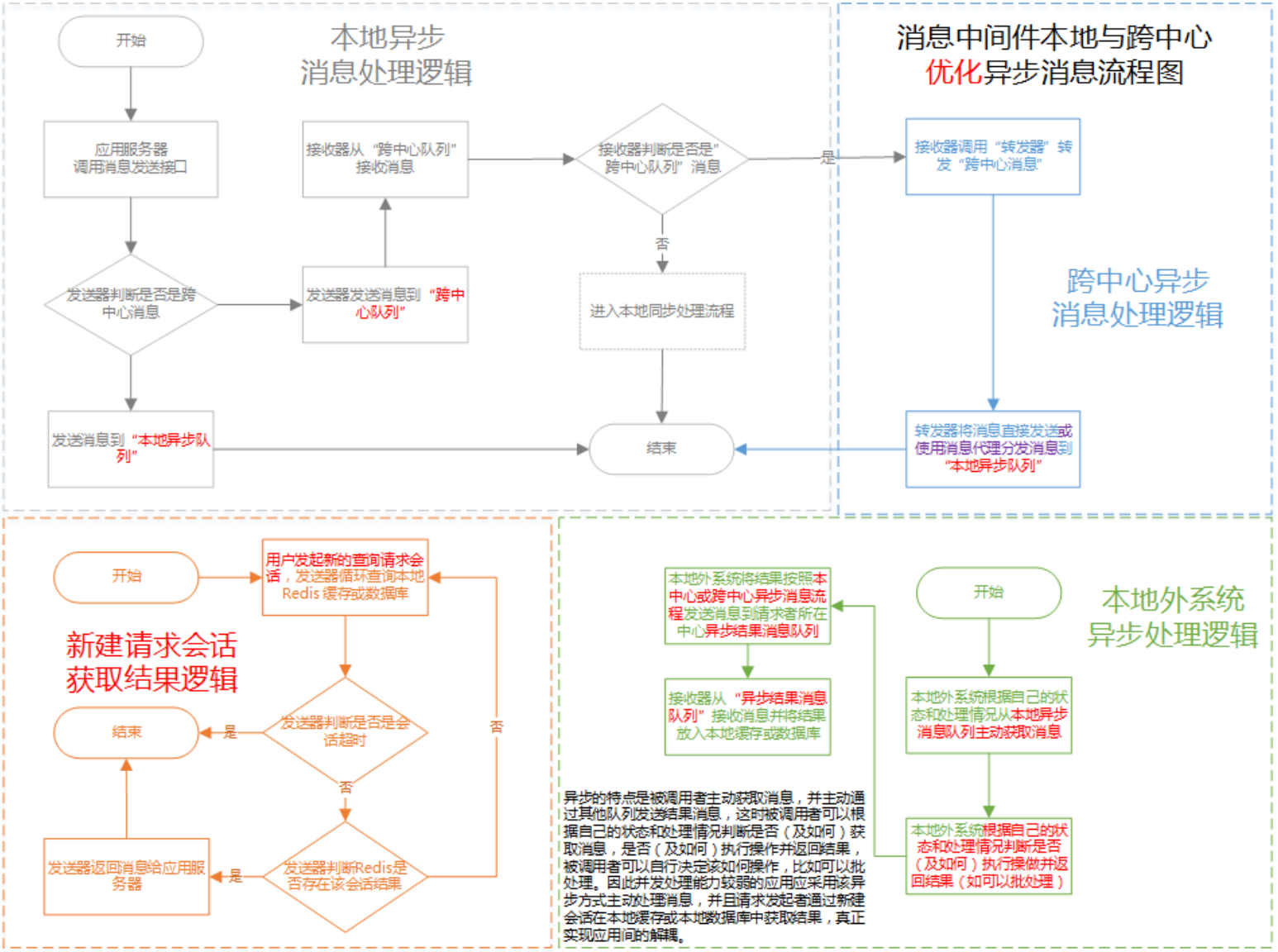


消息中间件本地与跨中心
同步消息流程图

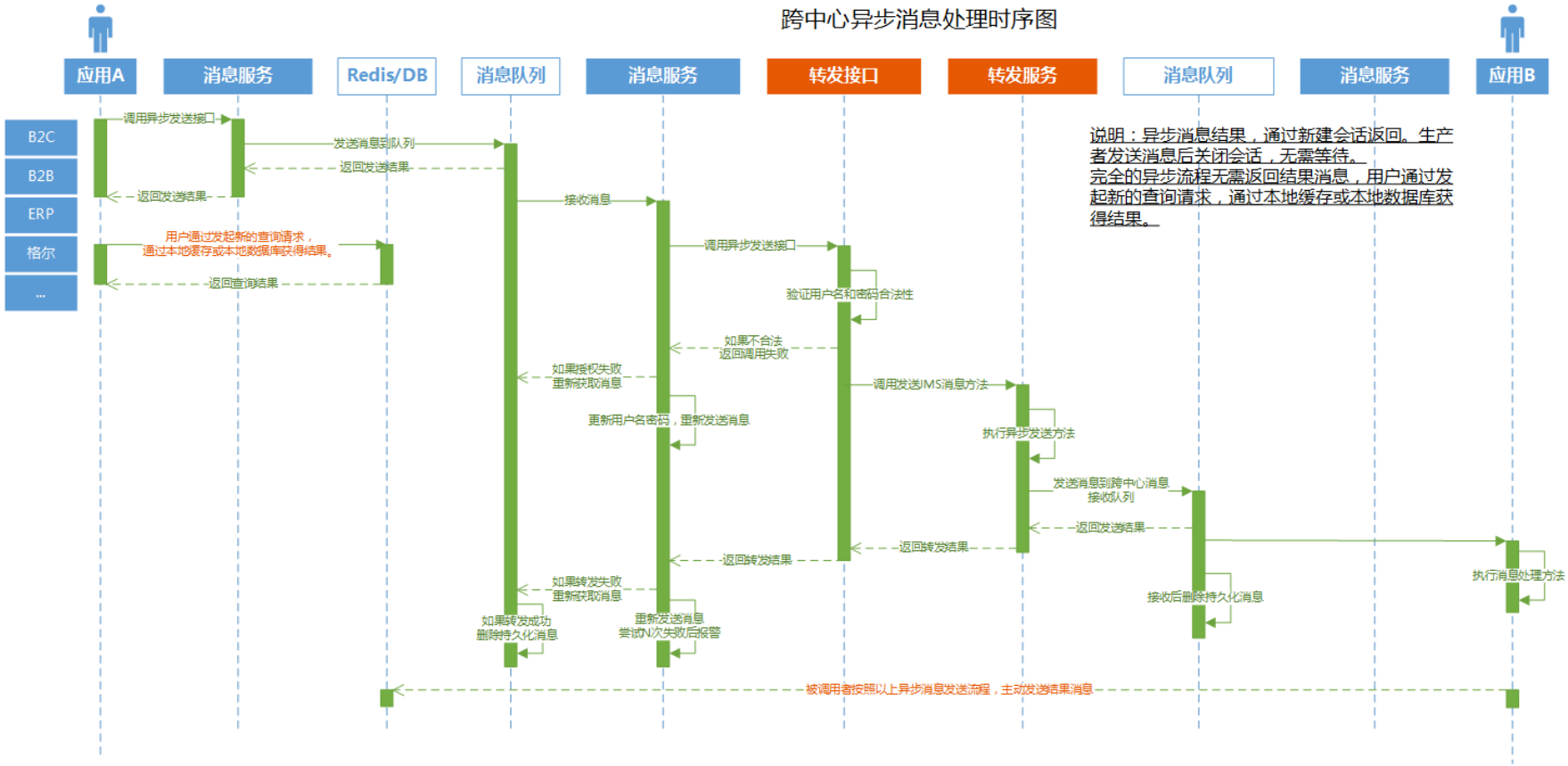
消息流程图-异步消息


开始
应用服务器
调用消息发送接口
发送器判断是否是跨
中心消息
发送消息到“本地异步队
列”
本地异步
消息处理逻辑
发送器发送消息到“跨中
心队列”
接收器从“跨中心队列”
接收消息
接收器判断是否是”
跨中心队列”消息
是
否
进入本地同步处理流程
结束
消息中间件本地与跨中心
异步消息流程图
接收器调用“转发器”转
发“跨中心消息”
转发器将消息直接发送或
使用消息代理分发消息到
“本地异步队列”
跨中心异步
消息处理逻辑
结果消息本地
缓存接收逻辑
发送器循环查询本地
Redis 缓存
发送器判断是否是会
话超时
是
否
结束
发送器判断Redis是
否存在该会话结果
是
否
发送器返回消息给应用服
务器
本地外系统将结果按照本
中心或跨中心异步消息流
程发送消息到请求者所在
中心异步结果消息队列
接收器从“异步结果消息
队列”接收消息并将结果
放入本地Redis 缓存
开始
本地外系统根据自己的状
态和处理情况从本地异步
消息队列主动获取消息
本地外系统根据自己的状
态和处理情况判断是否
（及如何）执行操做并返
回结果（如可以批处理）
本地外系统
异步处理逻辑
异步的特点是被调用者主动获取消息，并主动通
过其他队列发送结果消息，这时被调用者可以根
据自己的状态和处理情况判断是否（及如何）获
取消息，是否（及如何）执行操作并返回结果，
被调用者可以自行决定该如何操作，比如可以批
处理。因此并发处理能力较弱的应用应采用该异
步方式主动处理消息，并且请求发起者通过新建
会话在本地缓存或本地数据库中获取结果，真正
实现应用间的解耦。

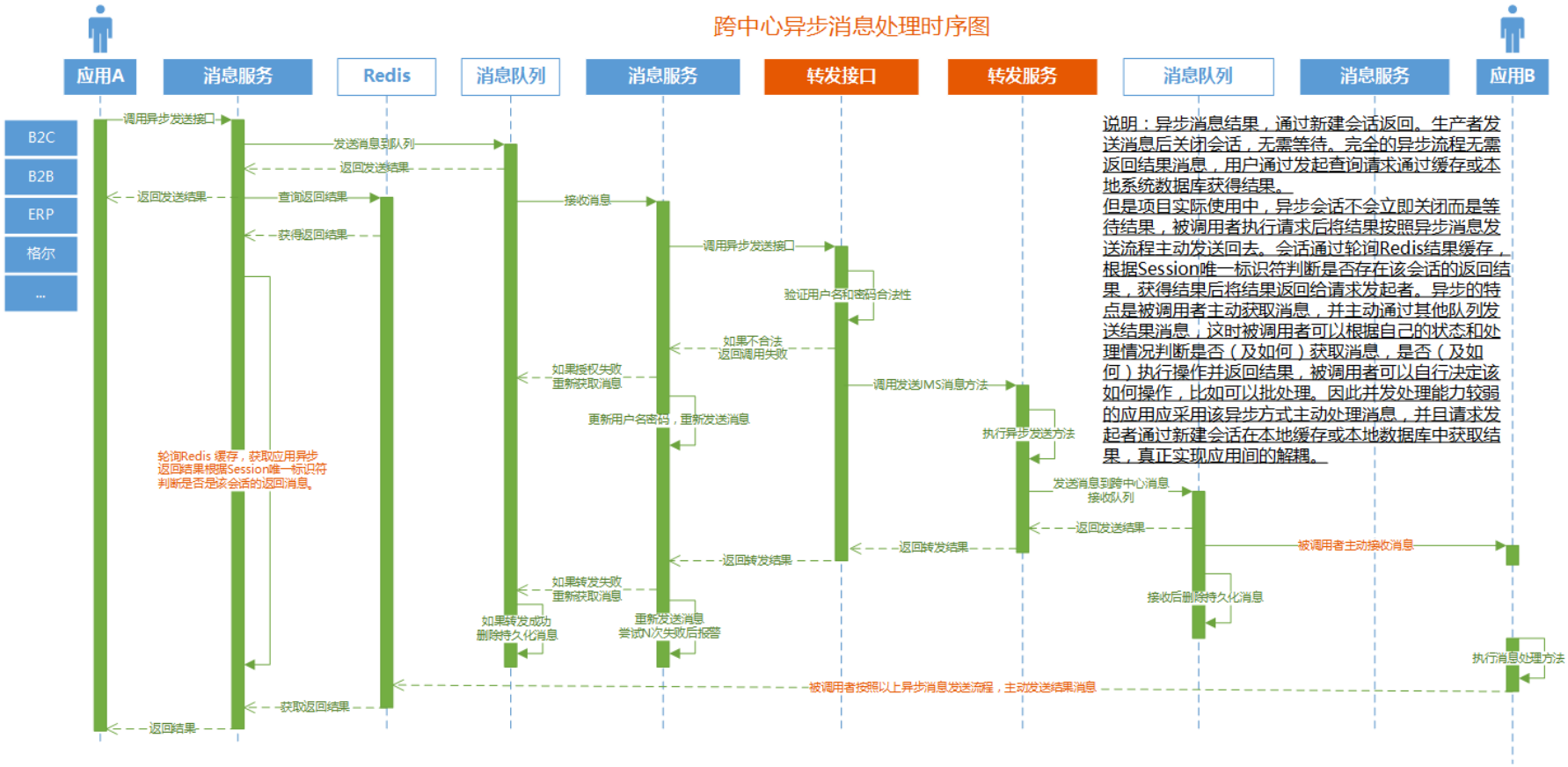
消息流程图-优化异步消息



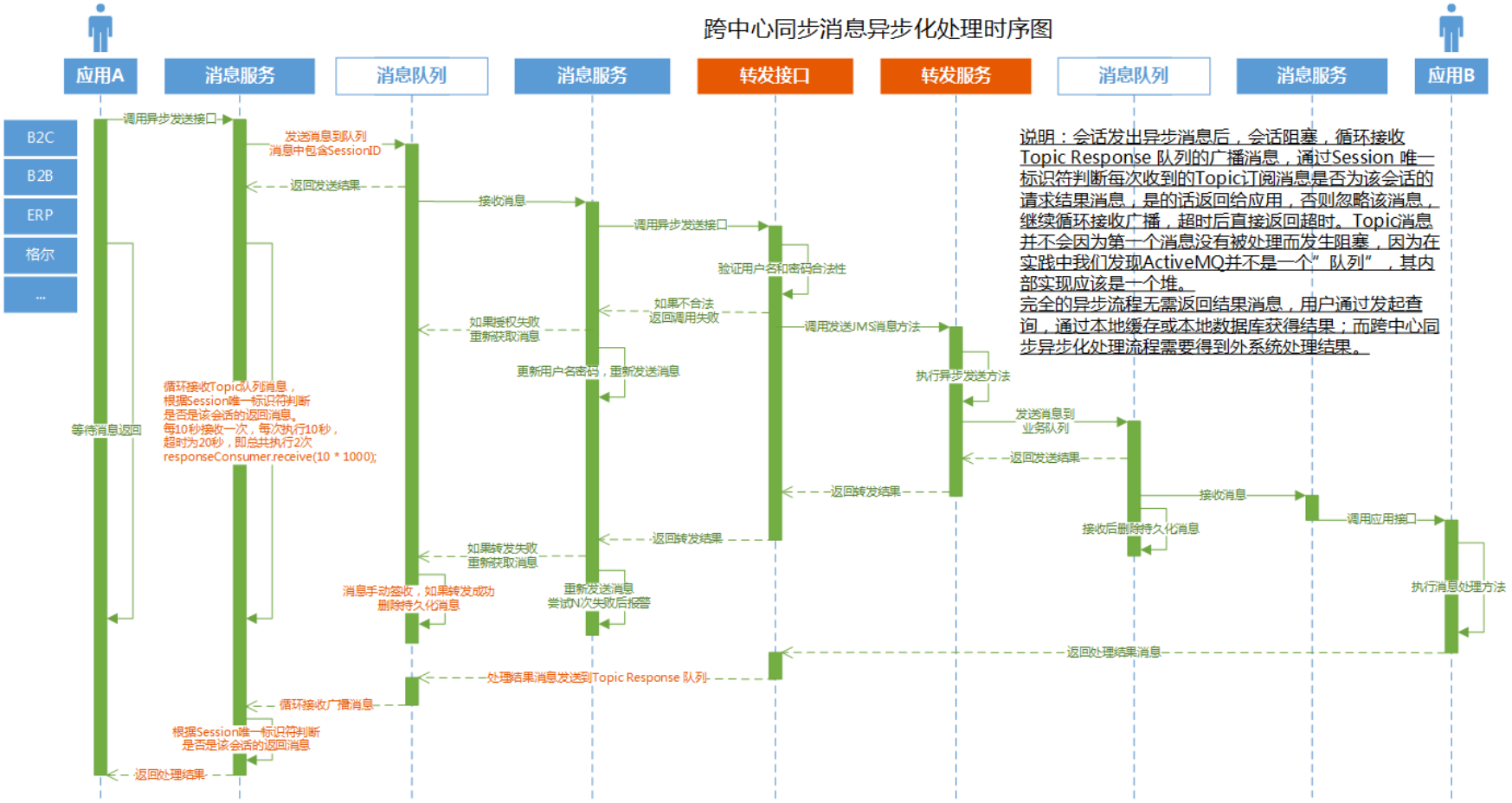
跨中心消息时序图-异步消息1



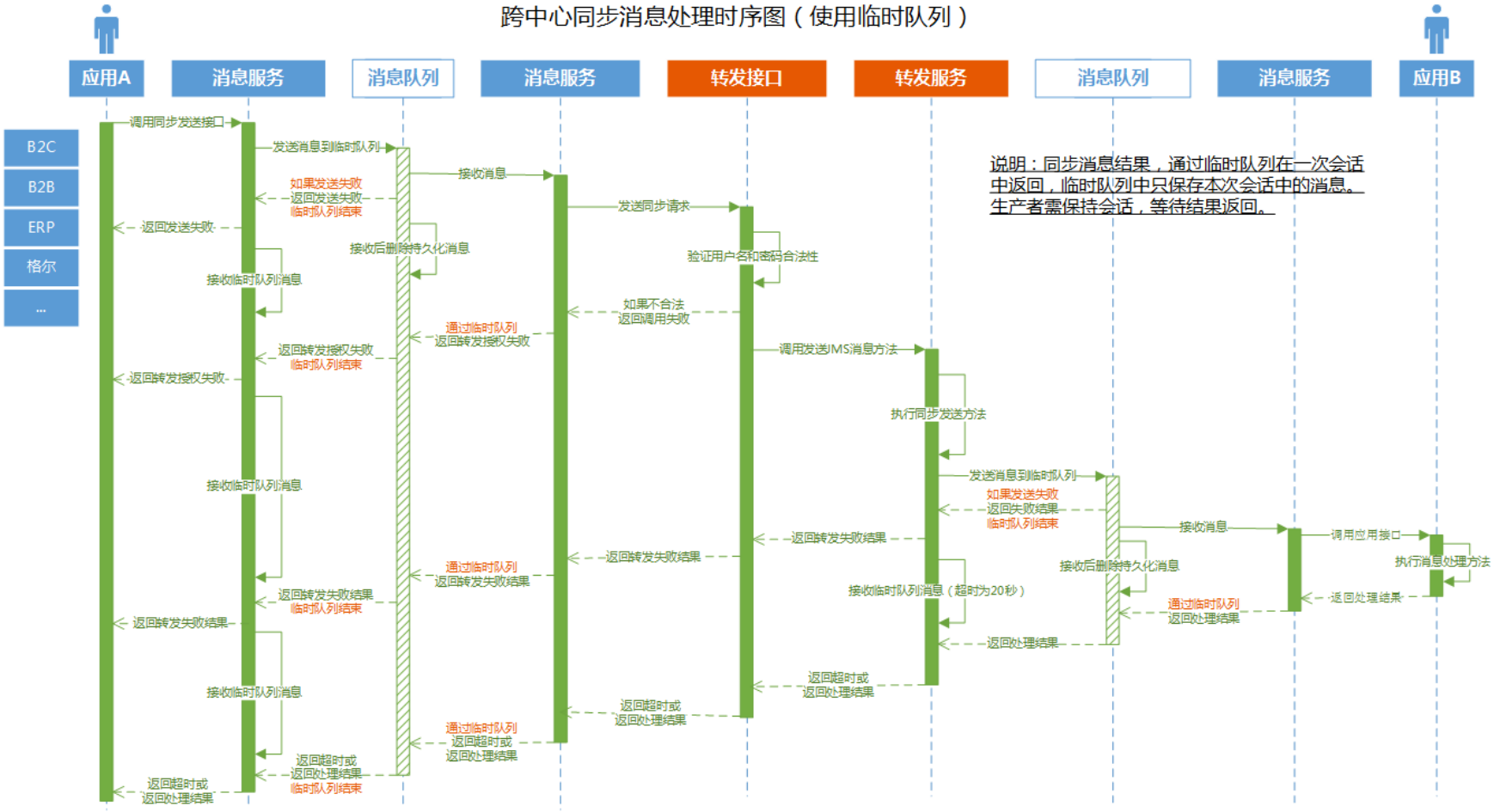
跨中心消息时序图-同步消息1



跨中心消息时序图-同步消息2



跨中心消息时序图-临时队列

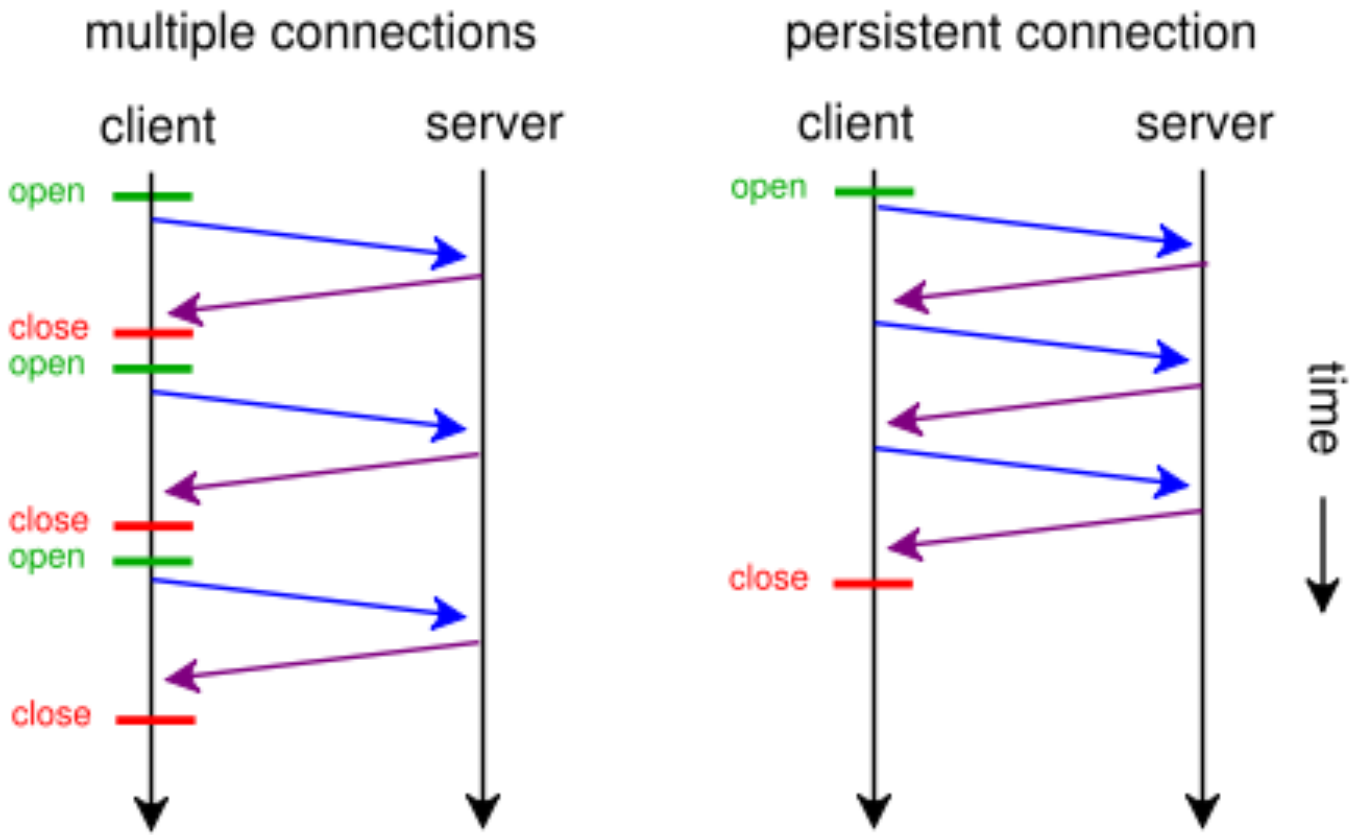


为什么不能使用临时队列

1）每一个消息生成一个临时队列，依然是并发访问没有产生“队列”的效果；
2）因为要通过临时队列维持每一个会话连接不断开，当大量会话同时到来时会造成网络拥堵，临时队列没有异步快速传递消息的效果；
3）同时也会造成一处堵塞，处处堵塞的火烧连营的效果。

异步消息的优势和特点

长连接，指在一个连接上可以连续发送多个数据包，在连接保持期间，如果没有数据包发送，需要双方发链路检测包。

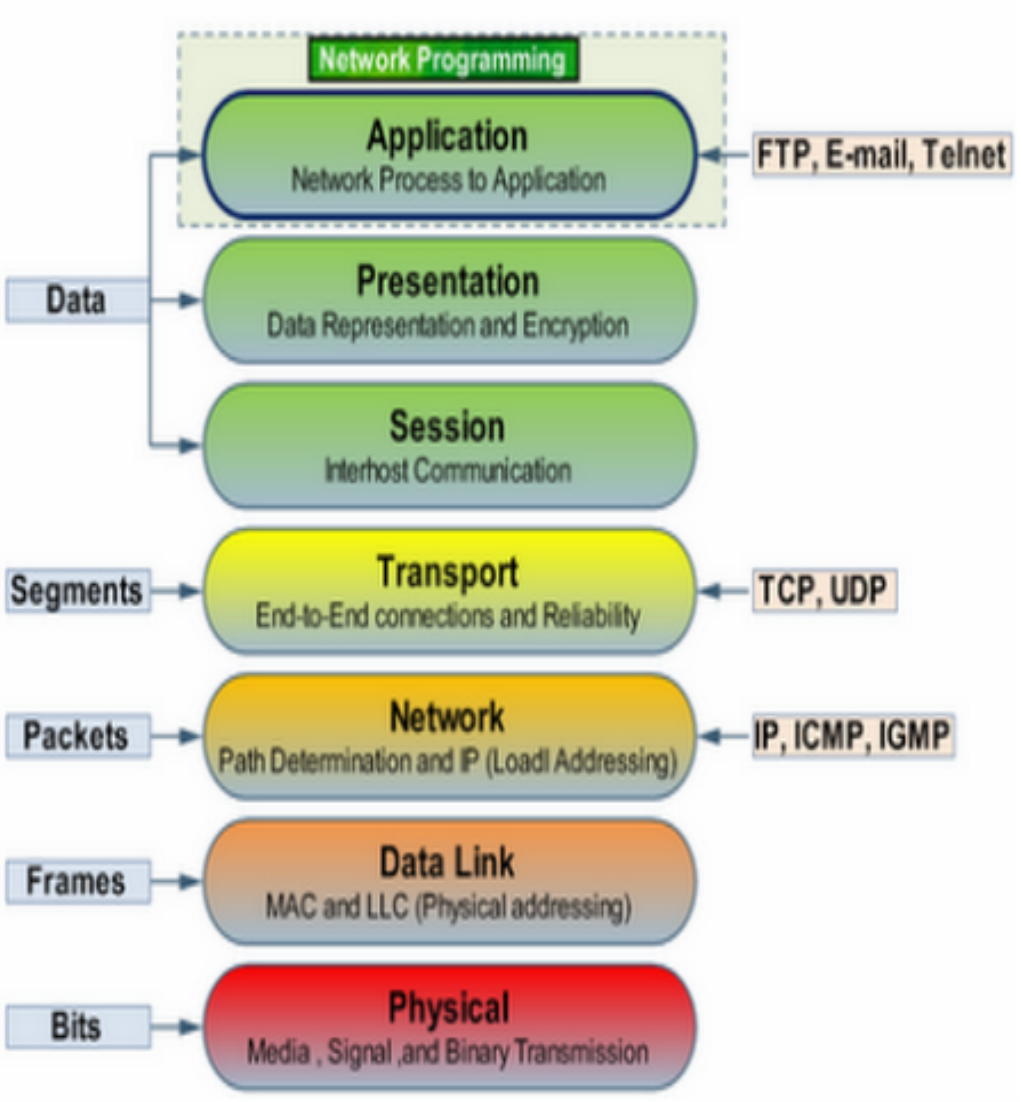


频繁短连接的TIME_WAIT问题

根据TCP协议定义的3次握手断开连接规定,发起socket主动关闭的一方 socket将进入TIME_WAIT状态，TIME_WAIT状态将持续2个MSL(Max Segment Lifetime)，在Linux下默认MSL为30秒,即60秒,TIME_WAIT状态下的socket才能被回收使用。

系统特性与协议模型

可用性
可靠性
安全性
……



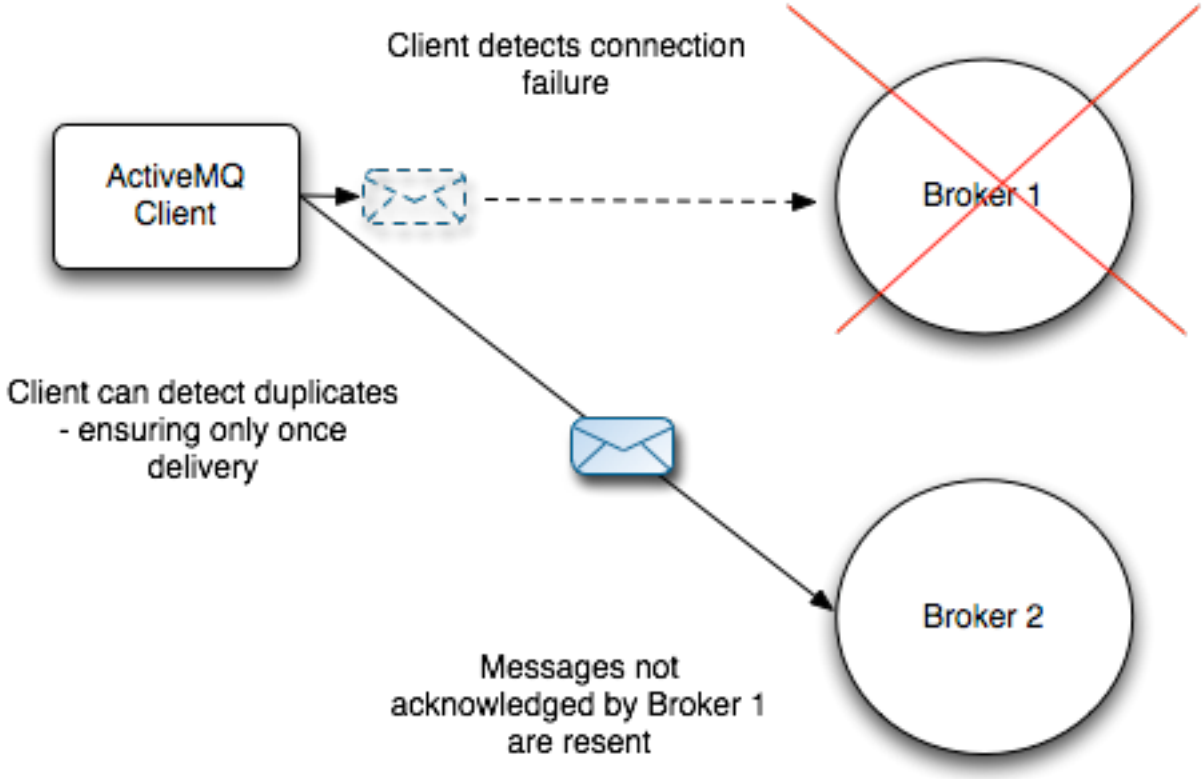
消息队列组件-ActiveMQ

- ✓ 高可靠性、事务性的消息队列
- ✓ 当前应用最广泛的开源消息中间件
- ✓ 2006年成为Apache项目
- ✓ Java代码44万行 (5.7.0)
- ✓ 支持多种接入协议和消息协议

消息队列组件-ActiveMQ

ActiceMQ 的可用性

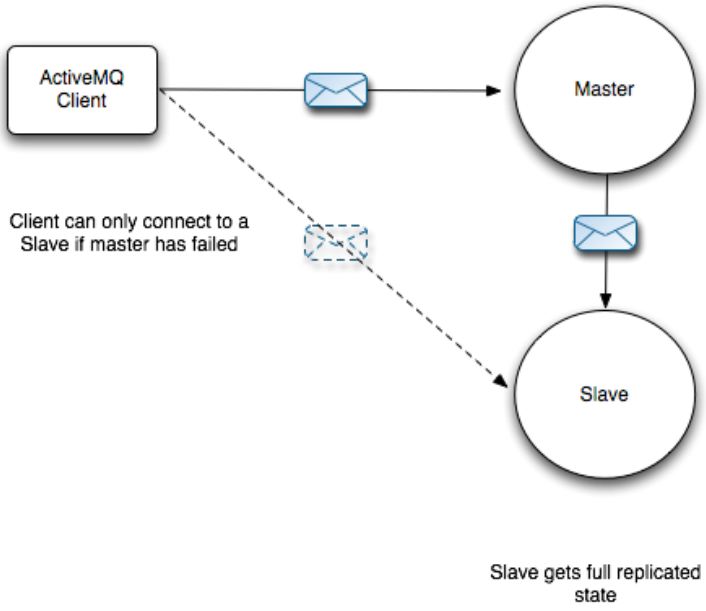
- ✓ Java和C++客户端支持无缝地failover 协议

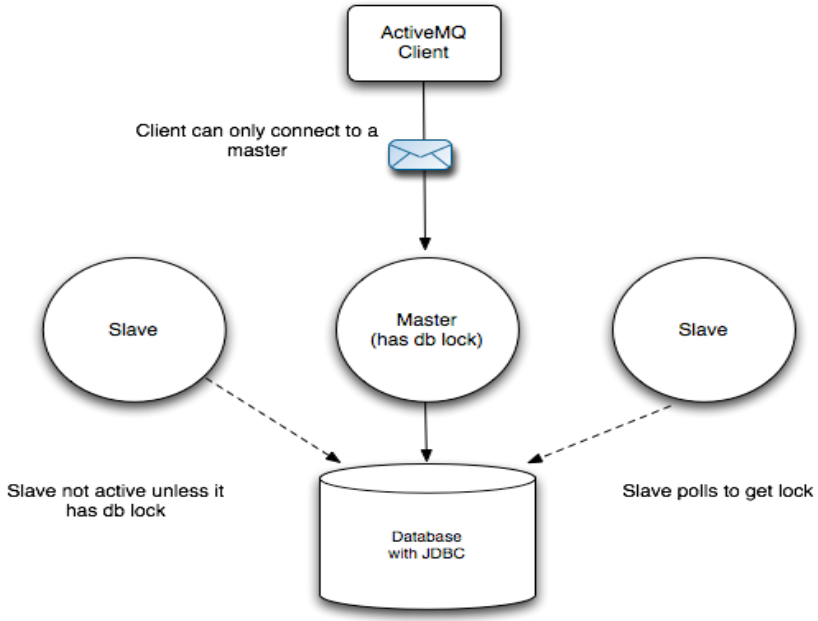


failover:(tcp://primary:61616,tcp://secondary:61616)

消息队列组件-ActiveMQ

基于failover 协议的主从配置

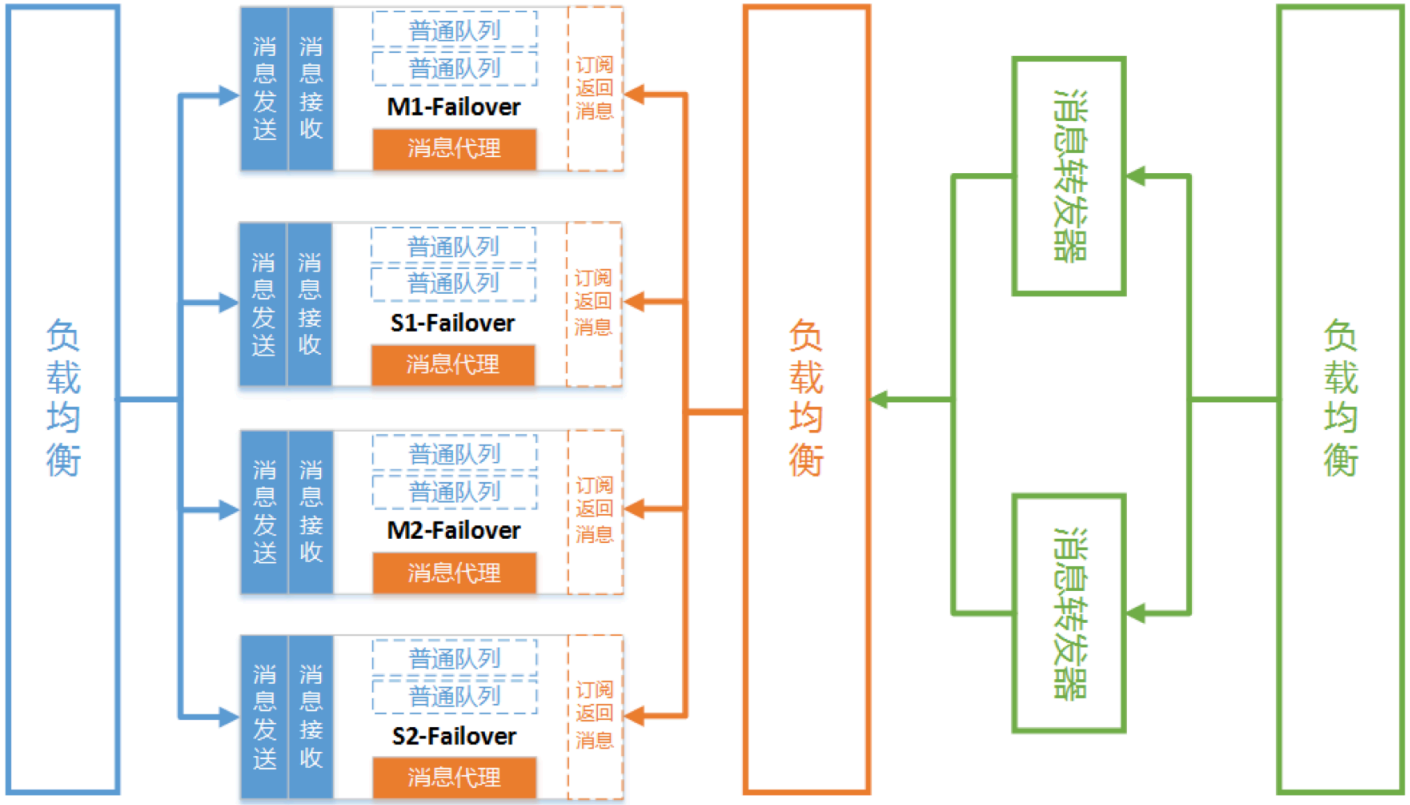




- ✓ 不共享存储，全复制（所有消息、所有应答、所有事务）
- ✓ 从服务器（备份服务器）默认不启动对外服务组件，只同步备份主服务器的所有数据

- ✓ 共享存储，从服务器个数没有限制，配置简单
- ✓ 从服务器一直待机状态，除非竞争到DB锁

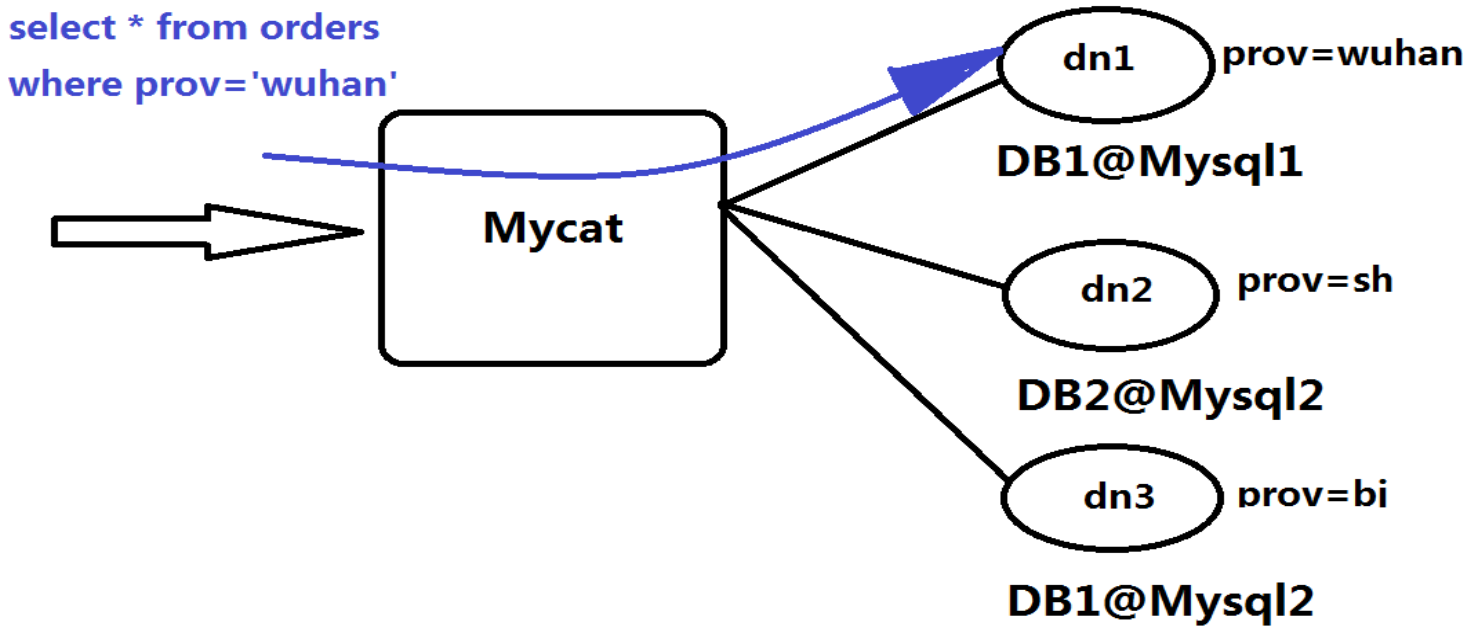
SunlightMQ 高可用与可扩容



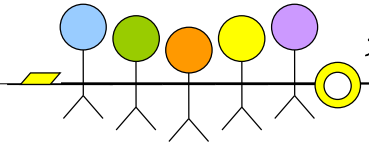
各组件本身都是无状态的，可以通过负载均衡部署架构进行扩容

SunlightMQ 消息存储扩容

在数据切分处理中，特别是水平切分中，中间件最终要的两个处理过程就是数据的切分、数据的聚合。选择合适的切分规则，至关重要，因为它决定了后续数据聚合的难易程度，甚至可以避免跨库的数据聚合处理。
拆分原则：
1. 避免或减少跨库join。
2. 选择最合适的拆分维度。



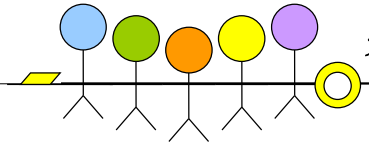
消息的可靠性



- 只有进行持久化才能保证消息的可靠性

- 需要在消息生产、存储、消费三个阶段保证消息的可靠性
  - ✓ 消息生产阶段（从生产者到中间件）：必须确认消息是否可靠到达中间件，准确返回结果（成功、出错、异常、超时等等）；
  - ✓ 消息存储阶段（从中间件到存储）：消息的可靠性依赖存储；
  - ✓ 消息消费阶段（从中间件到消费者）：
    - 消息处理完成之后再确认消息
    - 消息处理完毕的信号由应用层产生而不是网络层
    - 确认收到消息处理完毕的信号才能删除消息
    - 准确返回消息处理结果（成功、出错、异常、超时等等），尤其是异常处理

消息的事务性

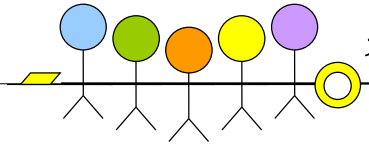


什么是消息事务性（ACID）？

更准确说是消息的“分布式”事务

- ✓原子性
- ✓一致性
- ✓隔离性
- ✓持久性

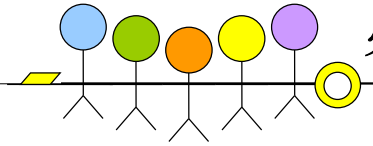
ActiveMQ与XA事务



JMS消息模型支持XA协议

- 在JMS的API中，以XA开头的接口，都是支持XA协议的接口（比如XAConnection、XASession）。
- XA事务的控制是在生产者、消息中间件、消费者三者之间的Session会话层的，因此这就要求业务操作的资源也要支持XA协议，这是一个比较大的限制。

基于组件架构的消息一致性



什么是消息一致性？

保证“业务操作” 与 “消息发送” 一致

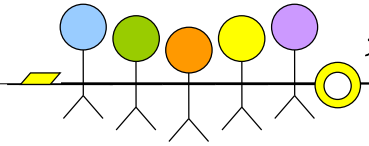
生产者消息一致性

- ✓ 操作完成 消息发出
- ✓ 操作失败 消息未发出

消费者消息一致性

- ✓ 处理成功 消息不再发送
- ✓ 处理失败 消息再次发送

生产者消息一致性



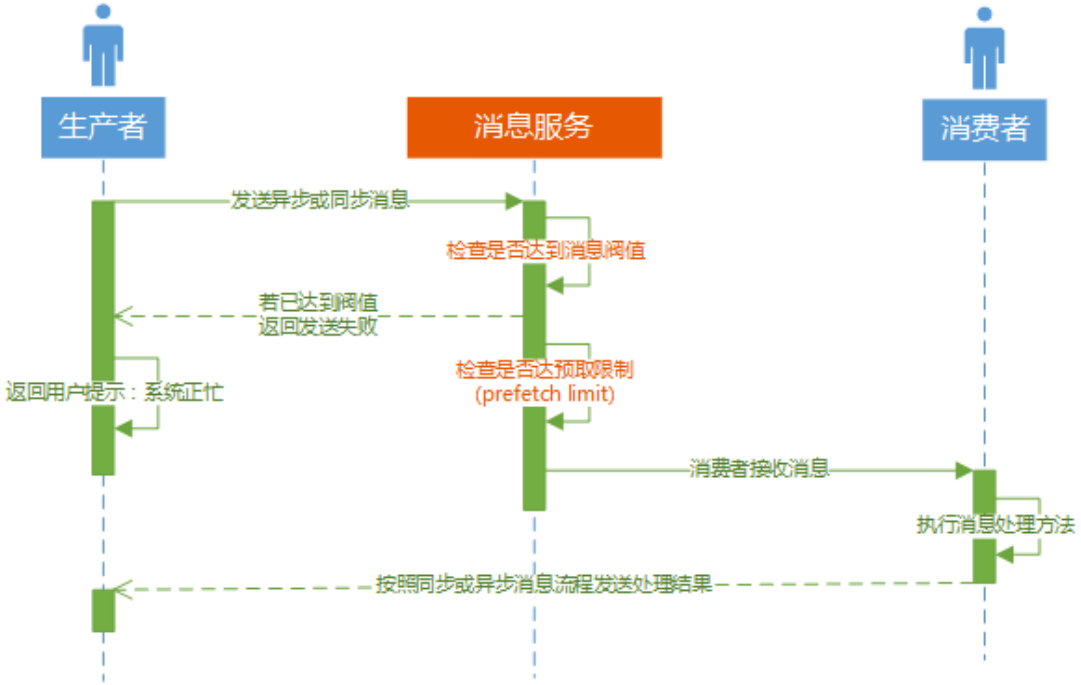
通过改变逻辑解决消息一致性问题

按照异步方式，改变原业务操作逻辑

原逻辑	改进后的逻辑
（1）执行操作	（1）发送消息给中间件
（2）发送消息给消息中间件	（2）消息中间件更新消息状态
（3）消息中间件入库消息	（3）执行操作
（4）消息中间件返回结果	（4）发送操作结果给中间件
	（5）消息中间件更新消息状态

真正发送消息之前，先进行消息发送确认

流量控制—消息阀值与预取限制

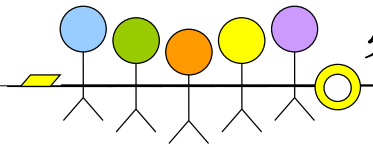


消息阀值：从5.x版本起，可以给每个生产者单独设置流控。流控简单的说就是控制生产者的在内存使用限制下的行为。当然流控的目的在于防止在将ActiveMQ作为内存MQ使用时，生产速度大于消费速度时将MQ撑爆的问题。

```
connectionFactory.setProducerWindowSize(1024000);
 //每个生产者最多发送约为1M消息
 //如果最多500个生产者，2G内存 = 2048 * 1024000 / 500  约 4M
```

预取限制(prefetch limit)：表示在某个时间段内，可能向消费者传输的最大消息量，如果达到该上限，那么停止发送，直到ActiveMQ收到消费者的acknowledgements(确认，表示已经处理了该消息)。prefetch limit可以针对每个不同的consumer来设置。

流量控制—Pull与Push方式对比



Pull 和 Push 方式对比

Push 方式

- 数据保存在服务器端
- 实时性高
- 服务器端需要依据消费者的处理能力做流控

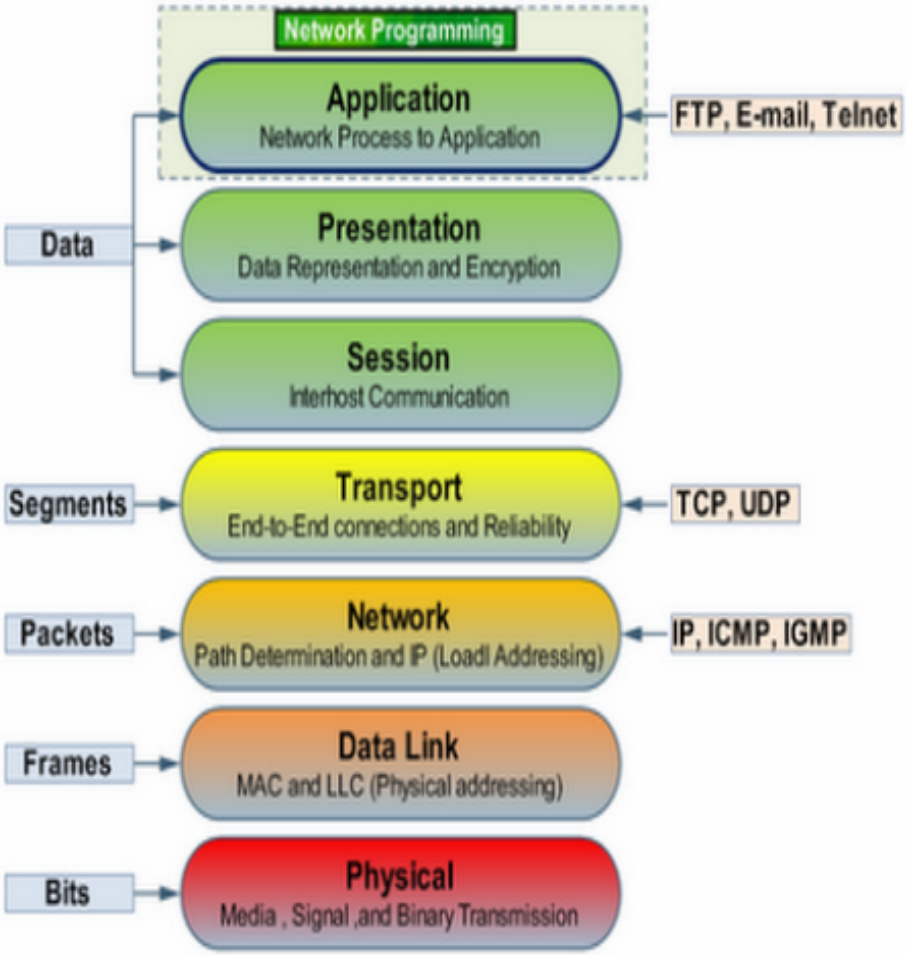
Pull 方式

- 数据保存在消费者端
- 实时性依赖与Pull的间隔时间
- 消费者可以根据自身的消费能力决定是否Pull 消息

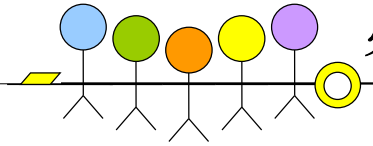
其他—压缩、编码、安全

- 应用层—消息压缩
  - ✓ Gzip
- 表示层—消息编码
  - ✓ Base64
- 会话层—消息安全
  - ✓ SSL
  - ✓ 连接用户名/密码

开源消息队列就像一块布，不能拿来就用，
我们要量体裁衣，把它变成一件衣服。

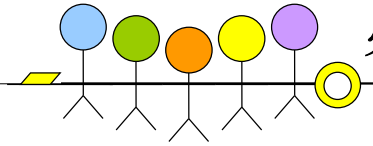


基于SunlightMQ的应用系统开发



使用SunlightMQ平台可以实现企业应用系统内部解耦和服务共享，提高团队敏捷度和开发效率，逐步使企业信息中心整体实现“软件即服务(SOA)”的设计思想。
软件即服务(SOA)的本质：是一个运用于企业内部一组应用而非单个应用的系统软件架构；重点关注网络和系统间的接口，而非实现；进行一切必要的标准化，确保流畅的交互和偶尔的重用；别无其他。

应用系统前端全部采用静态页面（HTML+JavaScript），使用HTTP+JSON作为与后台通讯的协议，SunlightMQ负责交易请求的转发。交易平台可以使用多种语言开发（C#/java/PHP/...）。应用系统前端部署时，不同模块的静态页面和JS分开部署，同时也可以通过统一门户将来自不同服务器的页面资源整合在一起。



亚马逊公司SOA软件设计开发原则

1、所有团队，要把他们的数据和软件功能通过服务接口对外公开；
2、团队之间的沟通只能通过这些接口；
3、不允许通过任何别的方式通讯：不能直接连接，走后门，不能直接读别的团队的数据，不能共享内存；
4、服务接口的后端的软件技术的选择，没有关系；
5、所有的服务接口设计时，都必须具备一个能力，允许日后让外界第三方开发者调用。没有任何例外；
6、不听话的人，将被开掉。

曾经在亚马逊工作了六年，2005年跳槽到谷歌的软件工程师，Steve Yegge 在2011年的一封公开信内介绍：

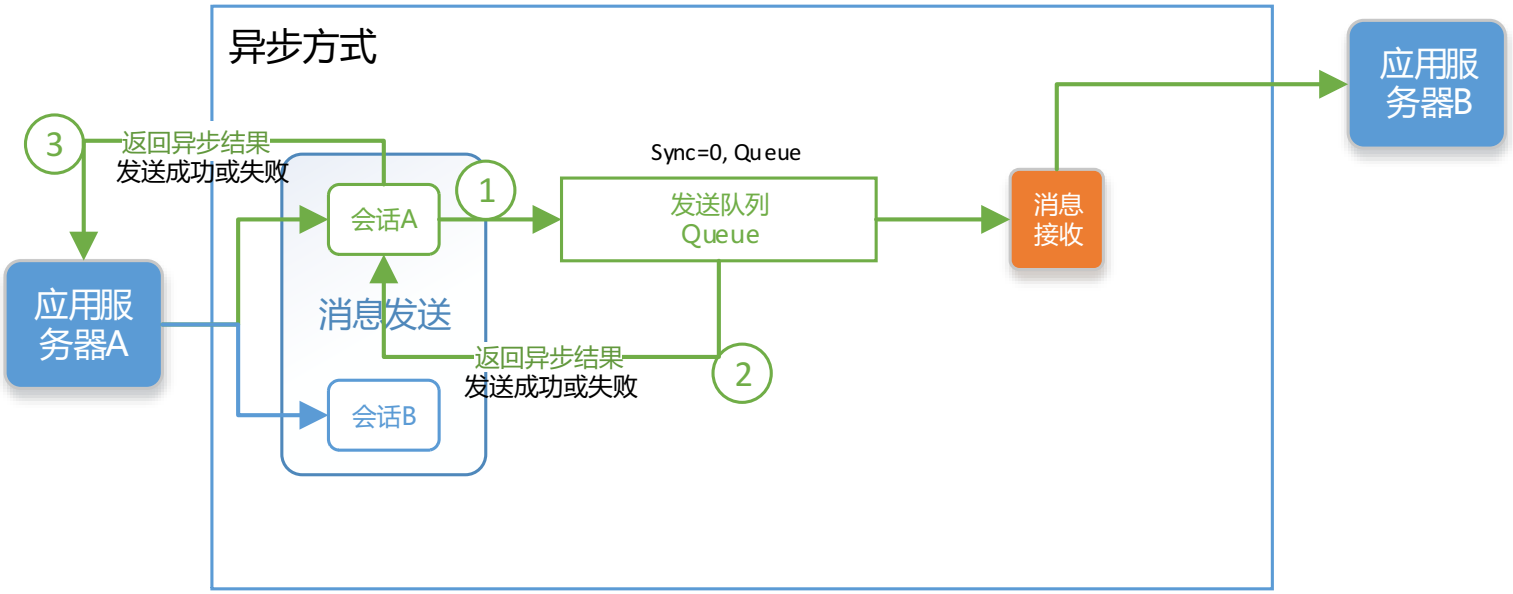
亚马逊的招聘流程本质上有缺陷。每个团队自己雇人，所以不同团队的员工招聘水准差别很大……，亚马逊的办公场所，污垢遍布，不花一分钱用于装饰……，亚马逊的程序代码一团糟，没有任何工程标准，完全看相关团队的选择……，也许有两百个不同的角度方式比较这两个公司，但除了三个以外，谷歌在所有方面都完胜亚马逊。

但是亚马逊有一件事，做得非常非常好，这远远弥补了亚马逊所有的政治上的，哲学上的和技术上的失误，2002年前后，贝索斯给所有员工发布了一个命令，关于内部软件设计的命令。

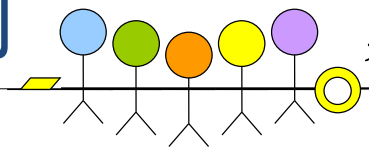
SunlightMQ 访问方式—异步访问

应用服务器A成功将请求发送到SunlightMQ平台后，即可关闭连接，不再等待接收应用服务器B的返回结果。

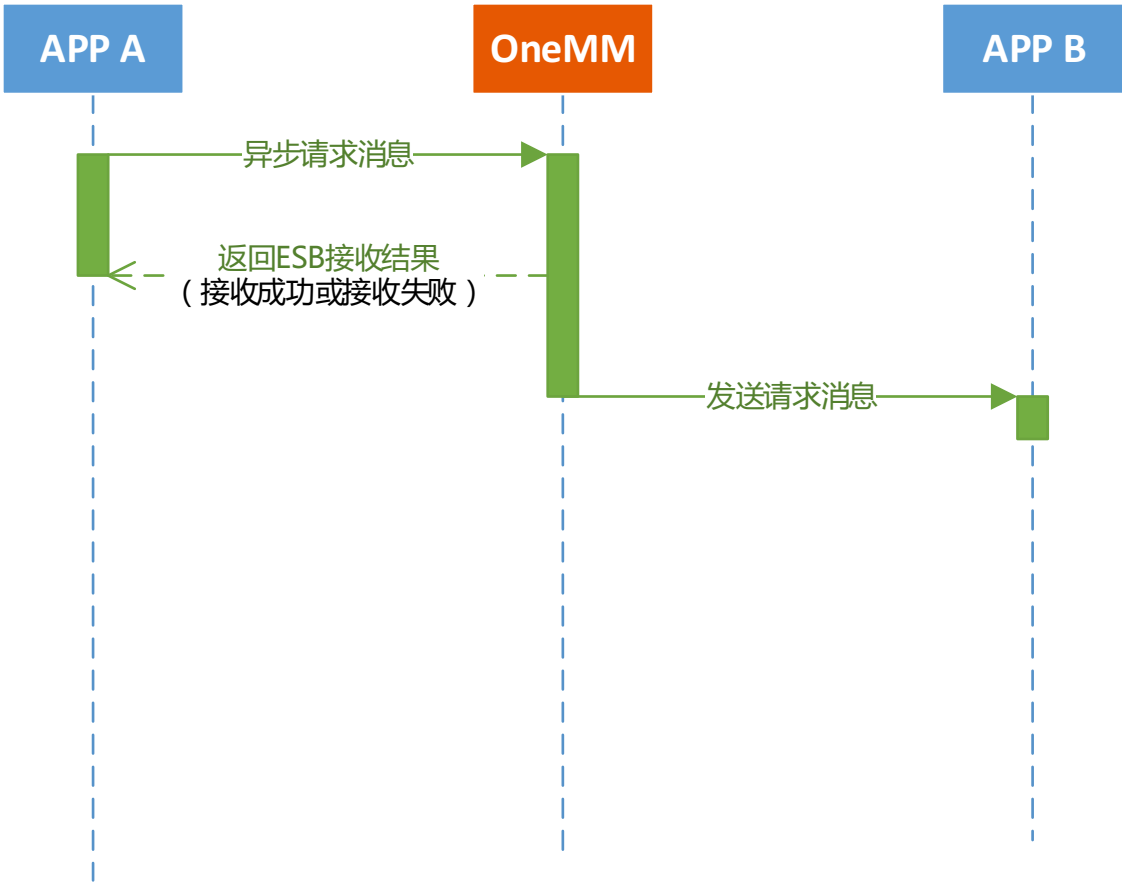
（1）消息流转组件图如下：



SunlightMQ 访问方式—异步访问



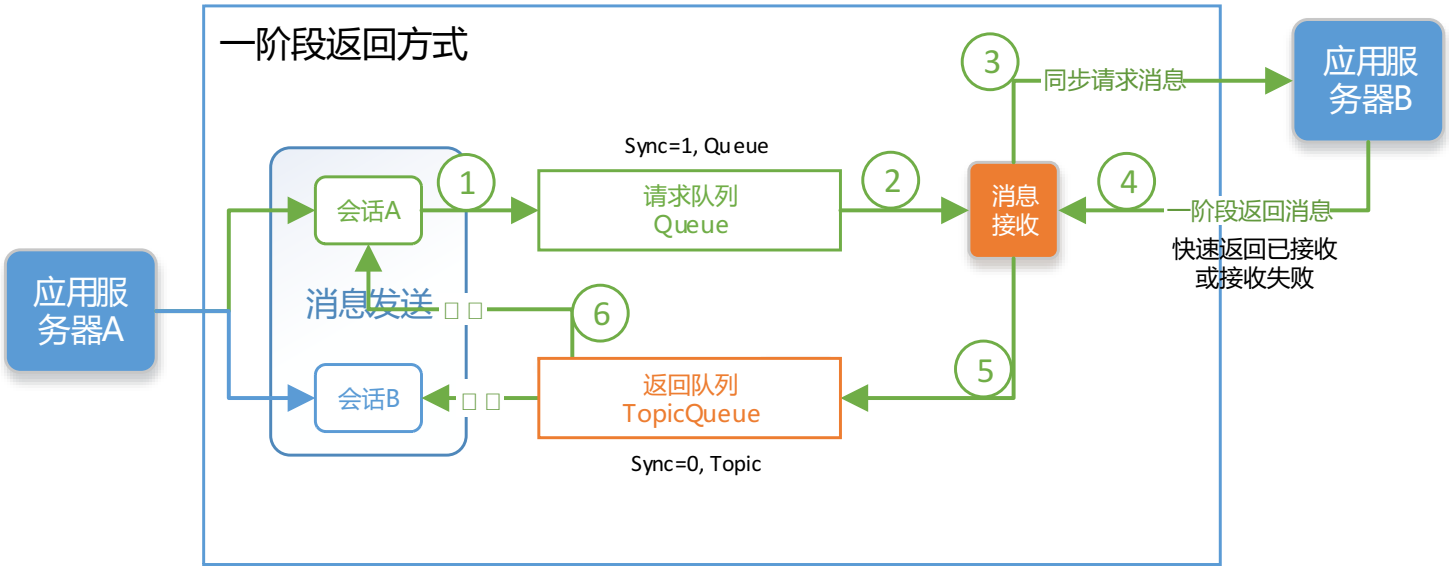
（2）消息流转时序图如下：



SunlightMQ 访问方式——一阶段返回

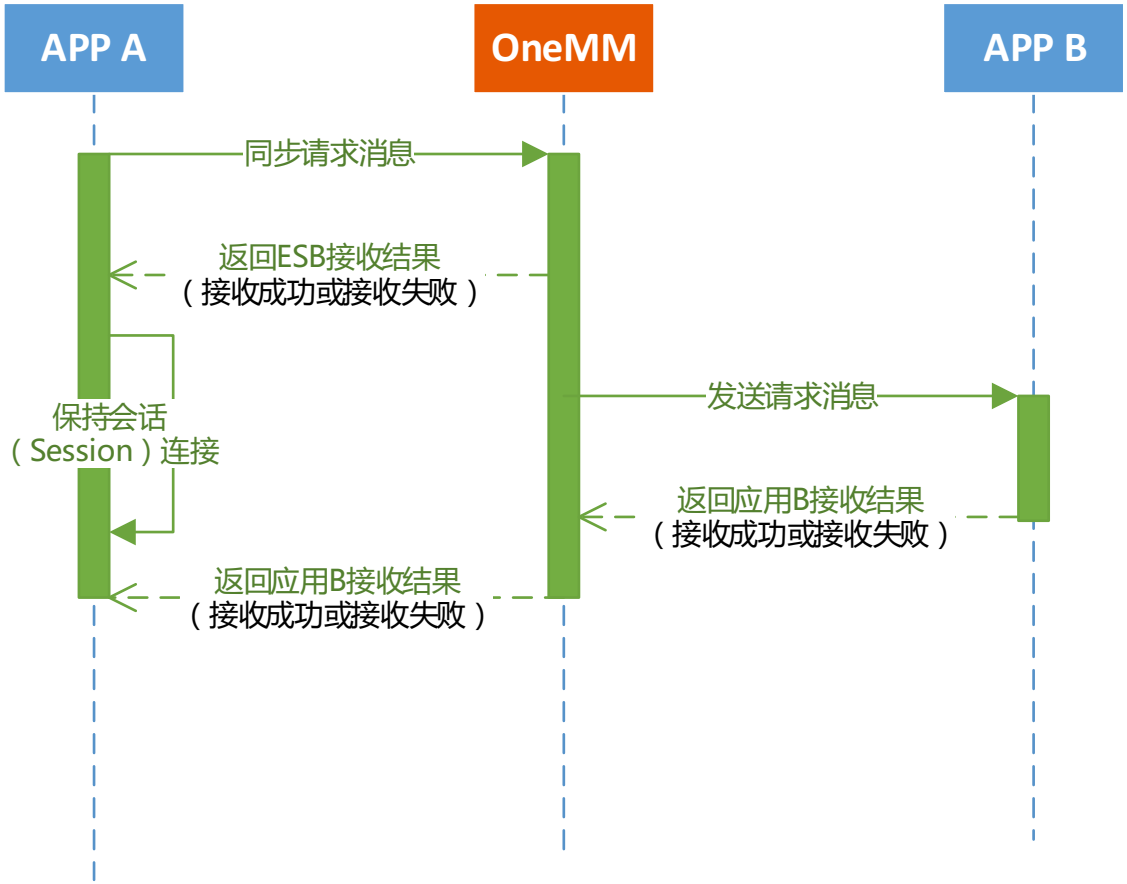
应用服务器A成功将请求发送到SunlightMQ平台后，保持连接，等待接收应用服务器B的返回结果。应用服务器B收到请求消息以后，直接返回接收结果：已接收或接收失败，不返回最终处理结果。

（1）消息流转组件图如下：

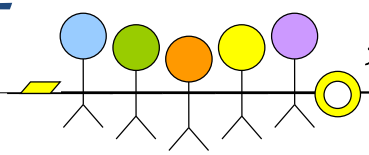


SunlightMQ 访问方式——一阶段返回

（2）消息流转时序图如下：



SunlightMQ 访问方式—二阶段返回



第一阶段：应用服务器A成功将请求发送到SunlightMQ平台后，保持连接，等待接收应用服务器B的返回结果。应用服务器B收到请求消息以后，直接返回接收结果：已接收或接收失败，不返回最终处理结果。
第二阶段：应用服务器B处理完成后，按照异步方式，通过“返回接口”将“最终处理结果”发送到“结果队列”，消息接收器调用应用服务器A的“接收接口”方法，返回结果给应用服务器A。

使用“二阶段返回方式”，应用服务器A需定义以下两个接口：

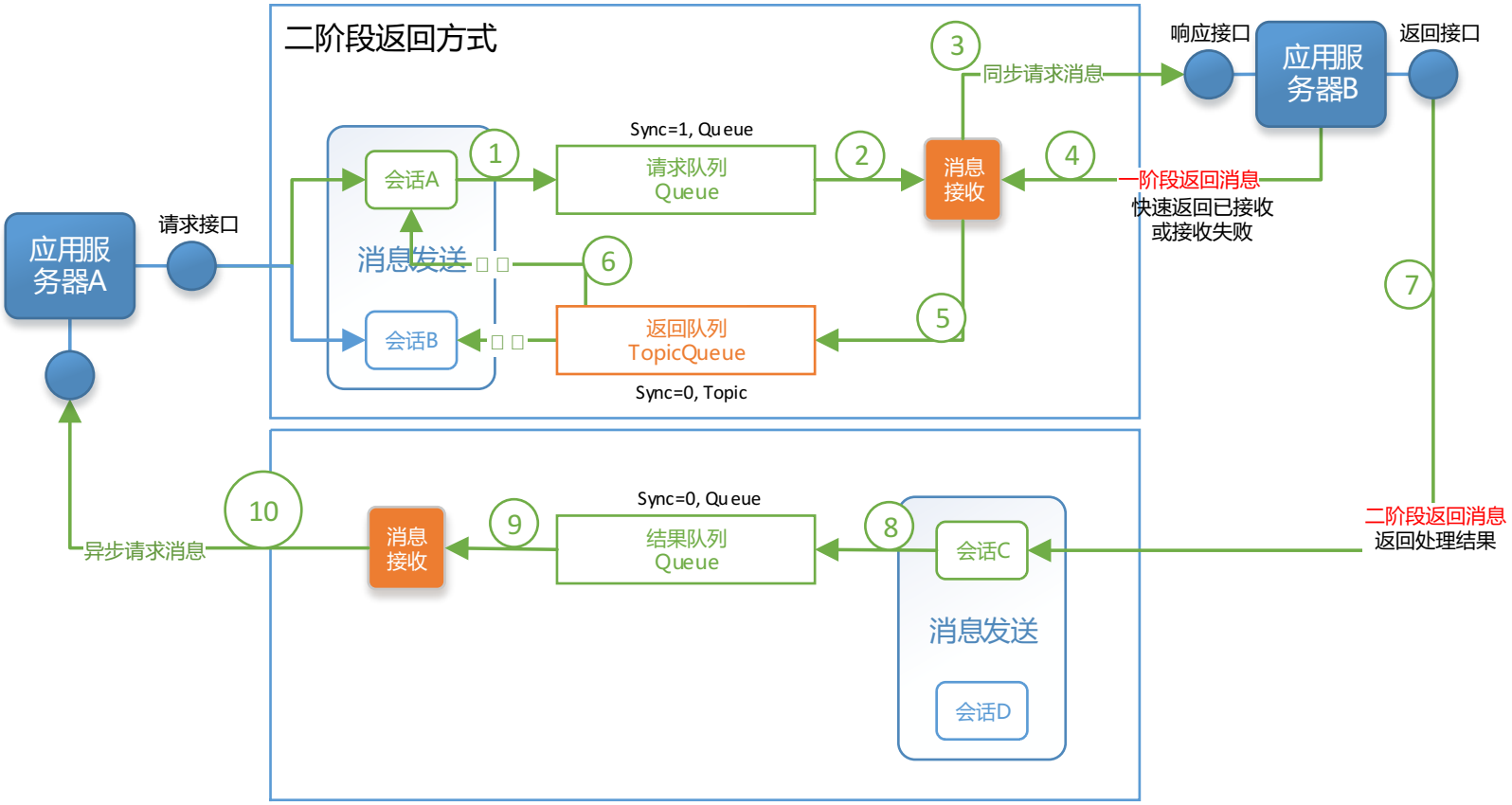
- 请求接口：请求业务服务
- 接收接口：接收处理结果

同时应用服务器B需定义对应的以下两个接口：

- 响应接口：处理业务请求
- 返回接口：返回处理结果

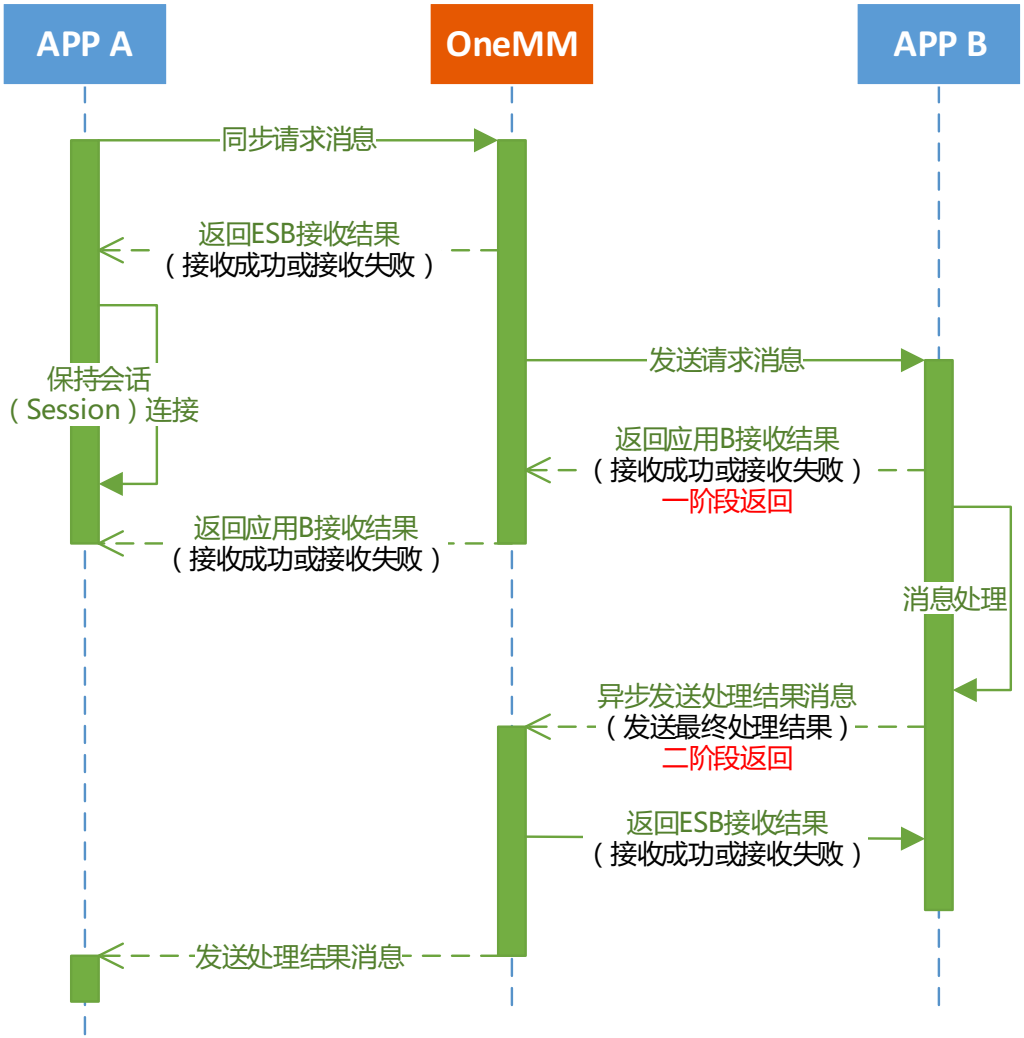
SunlightMQ 访问方式—二阶段返回

（1）消息流转组件图如下：



SunlightMQ 访问方式—二阶段返回

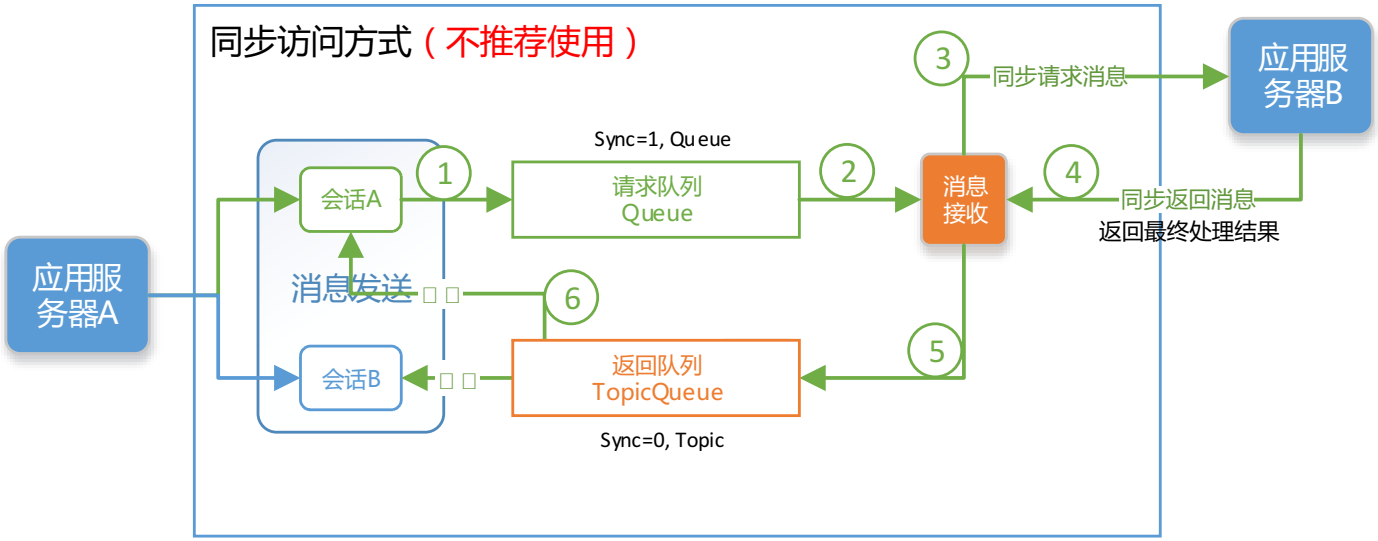
（2）消息流转时序图如下：



SunlightMQ 访问方式—同步访问

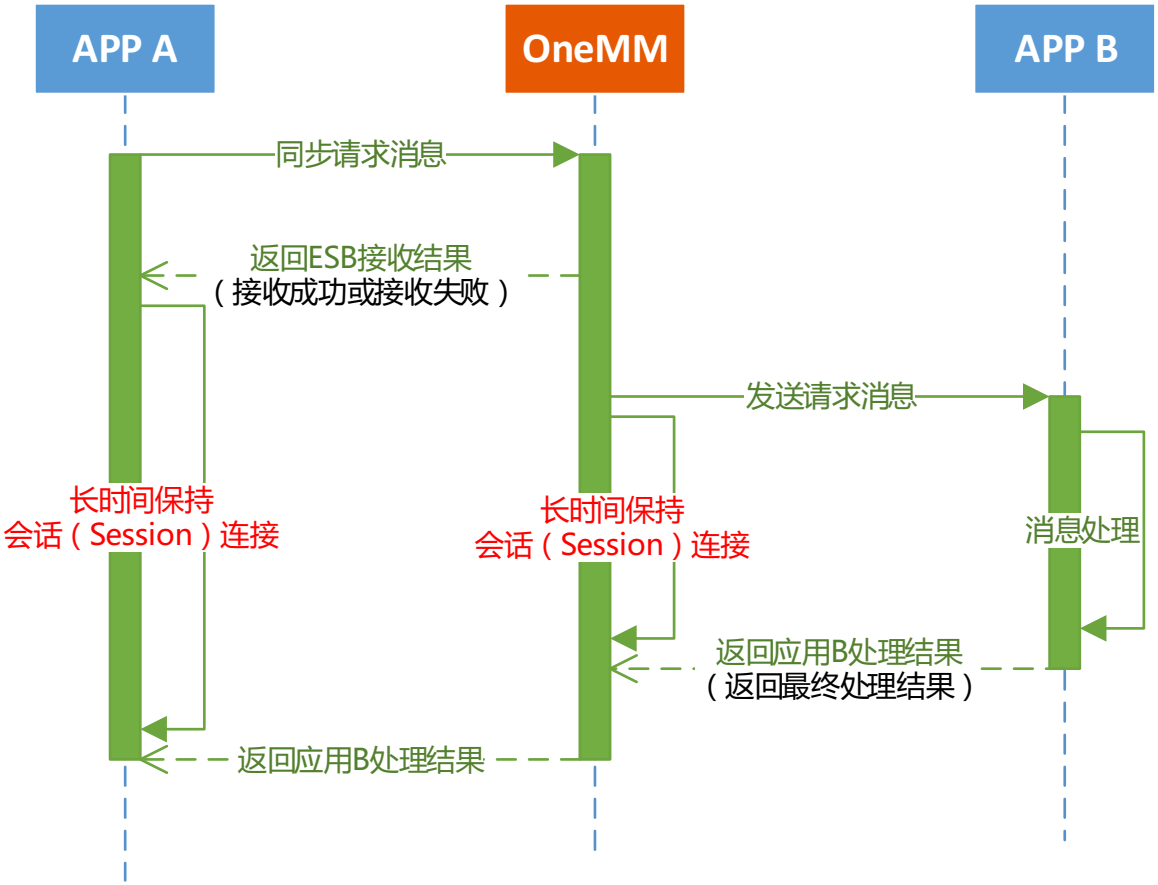
同步访问方式（不推荐使用）：应用服务器A成功将请求发送到SunlightMQ平台后，保持连接，等待接收应用服务器B的处理结果。应用服务器B收到请求消息以后，处理请求并返回最终处理结果。
同步访问方式会造成会话（session）长时间等待，高并发时影响系统整体稳定性，因此不推荐使用。需要返回最终处理结果的业务，推荐使用 “二阶段返回方式”。

（1）消息流转组件图如下：



SunlightMQ 访问方式—同步访问

（2）消息流转时序图如下：



其他开源消息队列-Apollo

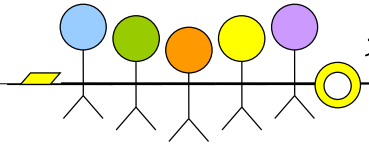
Apollo

- ✓ Apache ActiveMQ的改进版本
- ✓ Scala开发
- ✓ 多协议支持，不支持JMS
  - ✓ STOMP、AMQP、MQTT、OpenWire
- ✓ Message Swapping
  - ✓ 压缩内存消息，交互到磁盘

Apollo

ActiveMQ's next generation of messaging

其他开源消息队列-Kafka



Kafka

- ✓ Scala开发
- ✓ 高吞吐量是主要设计目标
- ✓ 基于ZooKeeper
- ✓ 消费者拉Pull 的模式，消费记录在消费者
- ✓ Consumer 分组
- ✓ 支持并行加载数据到Hadoop
- ✓ 适合处理日志类业务

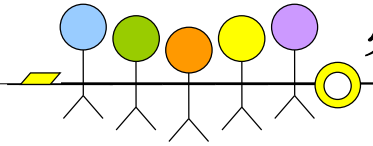
其他开源消息队列-RabbitMQ

RabbitMQ

- ✓ Erlang开发
- ✓ 性能好
- ✓ 持久化方式为异步到磁盘
- ✓ 工具和插件丰富
- ✓ 集群不支持动态扩展

RabbitMQ™
Messaging that just works

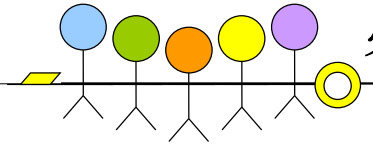
其他开源消息队列-ZeroMQ



ZeroMQ

- ✓ ZeroMQ是一个并发框架做的socket库（是一个传输层API库），通过引用ZeroMQ程序库，在应用之间发送消息，任何应用程序节点都可作为一个MQ
- ✓ 低延时，高性能，最高每秒43万条消息
- ✓ 非持久性队列，宕机后数据将会丢失

业务场景与消息队列选型



场景1：注重消息可靠性，关键数据的异步处理

- ✓ ActiveMQ

场景2：注重性能，可靠性要求不高

- ✓ ZeroMQ
- ✓ Rabbitmq

场景3：注重吞吐量，如日志收集

- ✓ Kafka
- ✓ MetaQ（Kafka java实现）

没有完美的产品或解决方案，按照场景选择合适的产品并根据特殊需求加以改造

感谢您的观赏！

我的联系方式

微信：life3721com
邮件：1526330277@qq.com

互联网开发无小事，因为互联网应用要面对海量信息，一个小小的问题也能被无限放大，就像汪洋中的蝴蝶效应。
一个互联网应用遇到的问题可能与系统、网络、应用、架构、数据等各个层面的问题有关，只有耐心发现并处理好每一个相关层面的问题，才能最终形成一款好的互联网应用。时刻提醒自己当下一波巨浪到来前我们会准备的更好！