



# 准备好发射了吗？

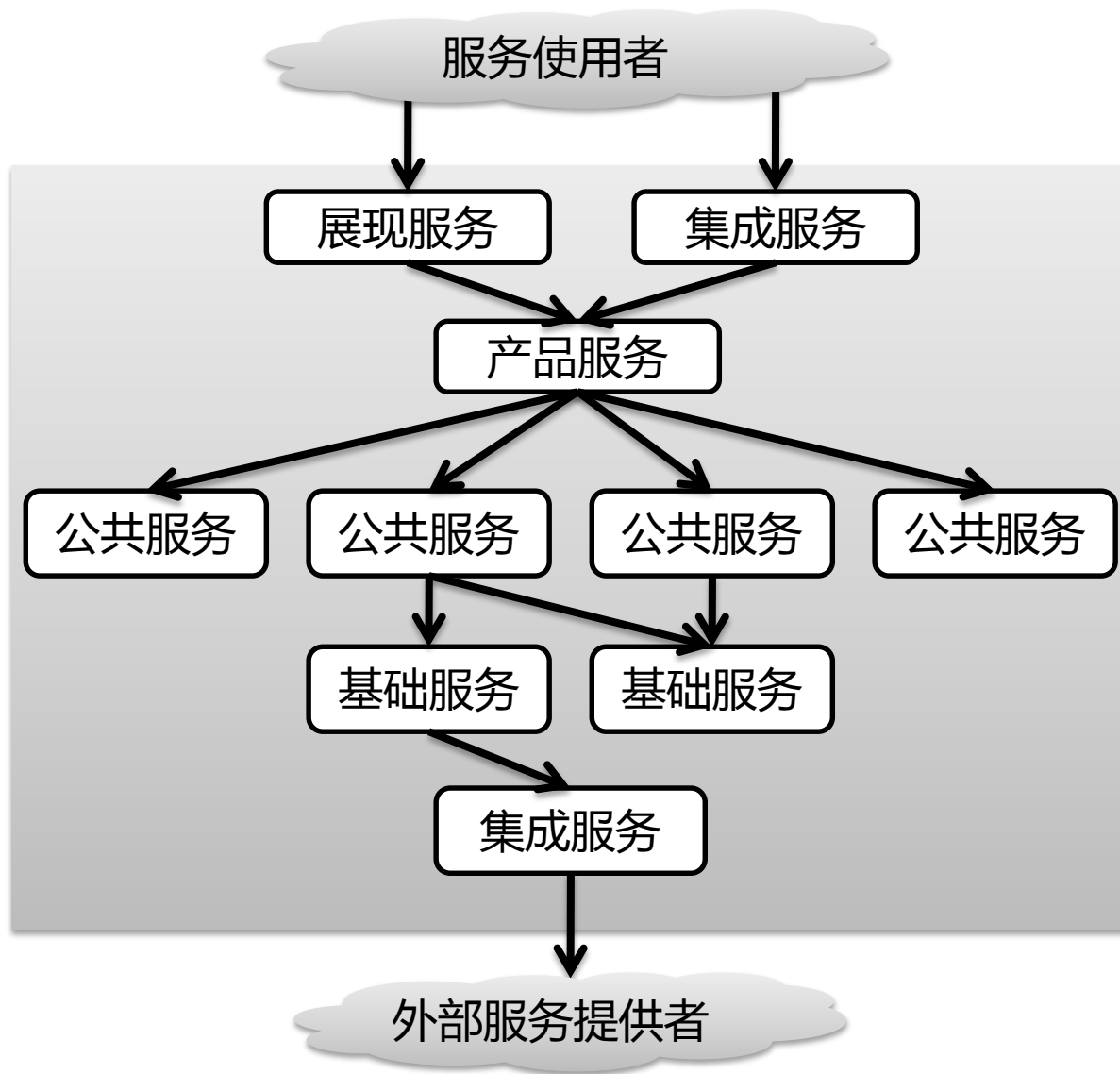
## 面向生产环境的SOA系统设计

程立  
Alipay.com

IT168系统架构师大会，北京，2009年8月28日

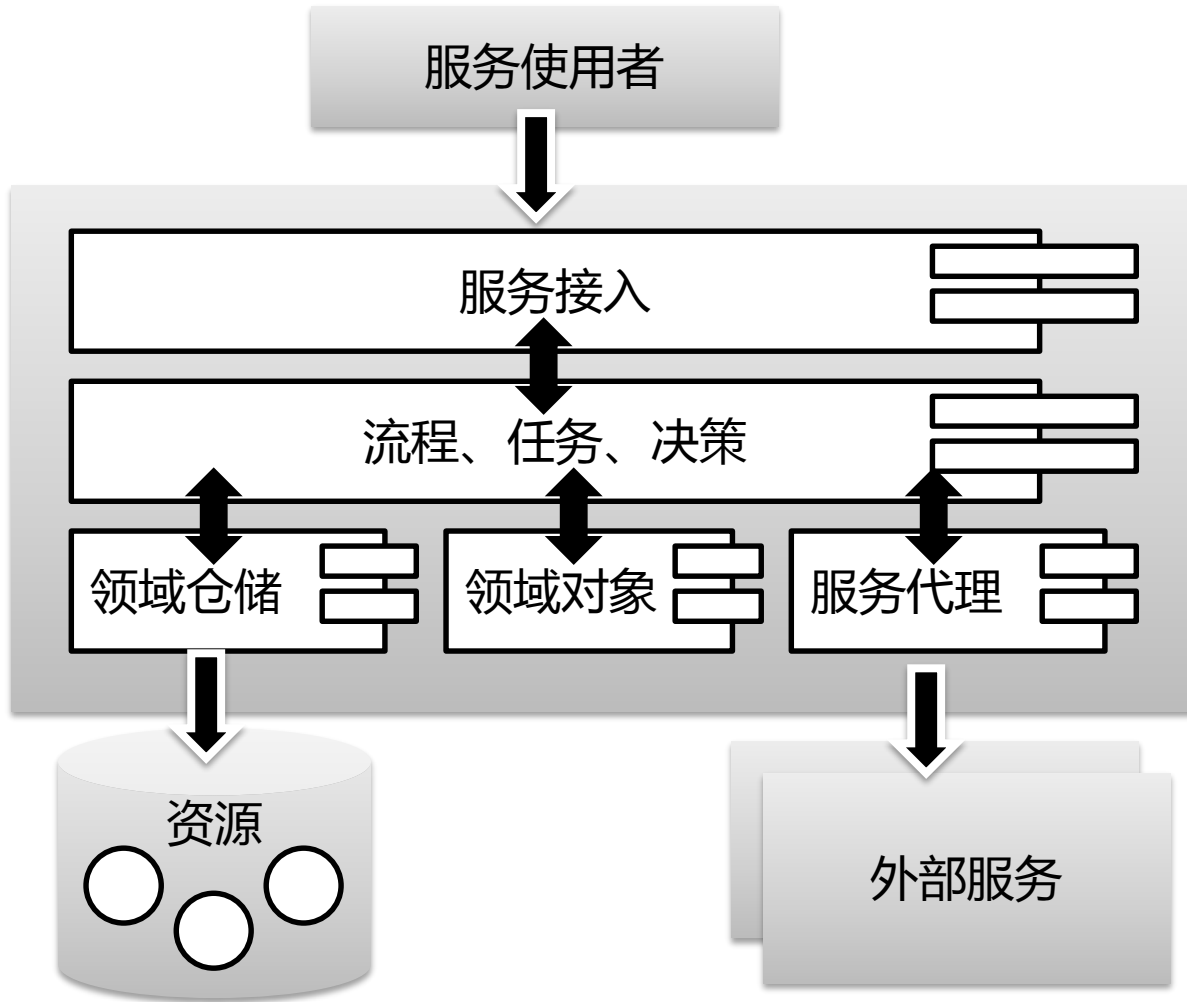
# 典型SOA应用

一个SOA应用  
由一系列服务  
松散复合而成。



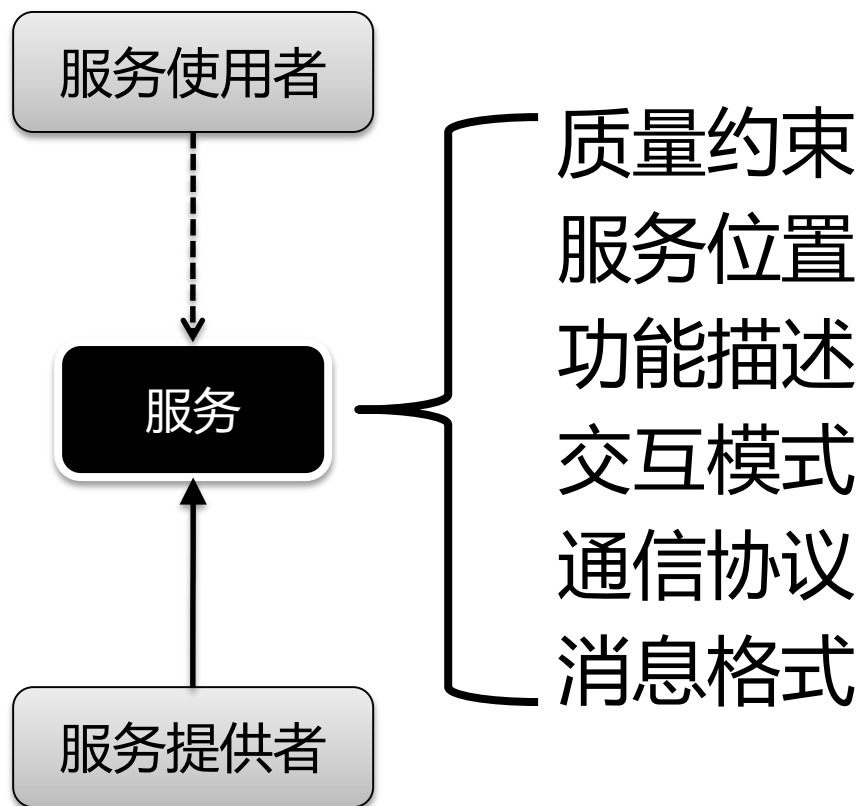
# 服务的内部

每个服务都是自包含、自主运行的功能单元。



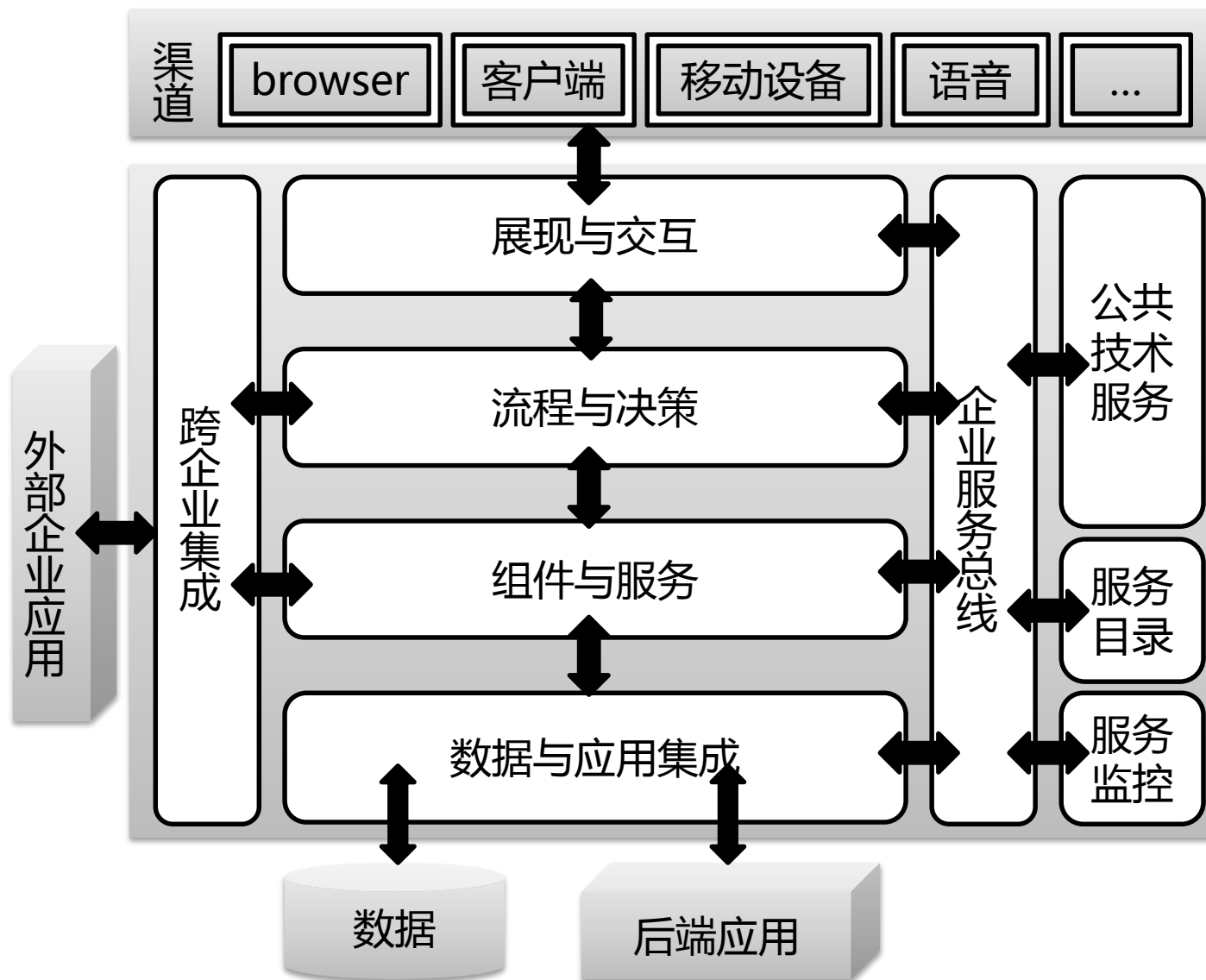
# 服务是基础架构单元

作为基础业务、应用与技术架构单元，服务具有丰富的含义。



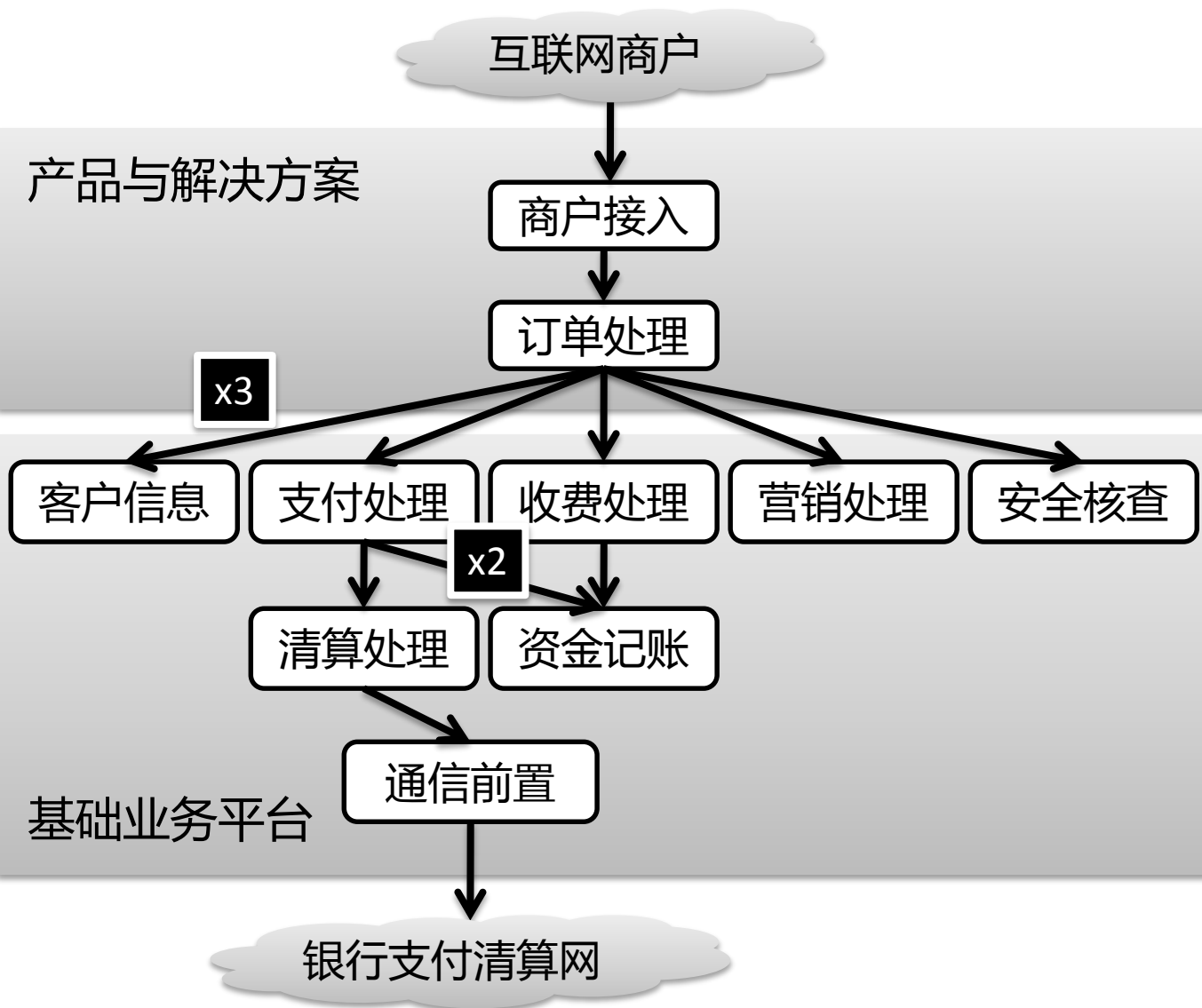
# SOA技术基础设施

SOA应用需要一系列技术基础设施的支持。



# 一个典型的电子支付应用

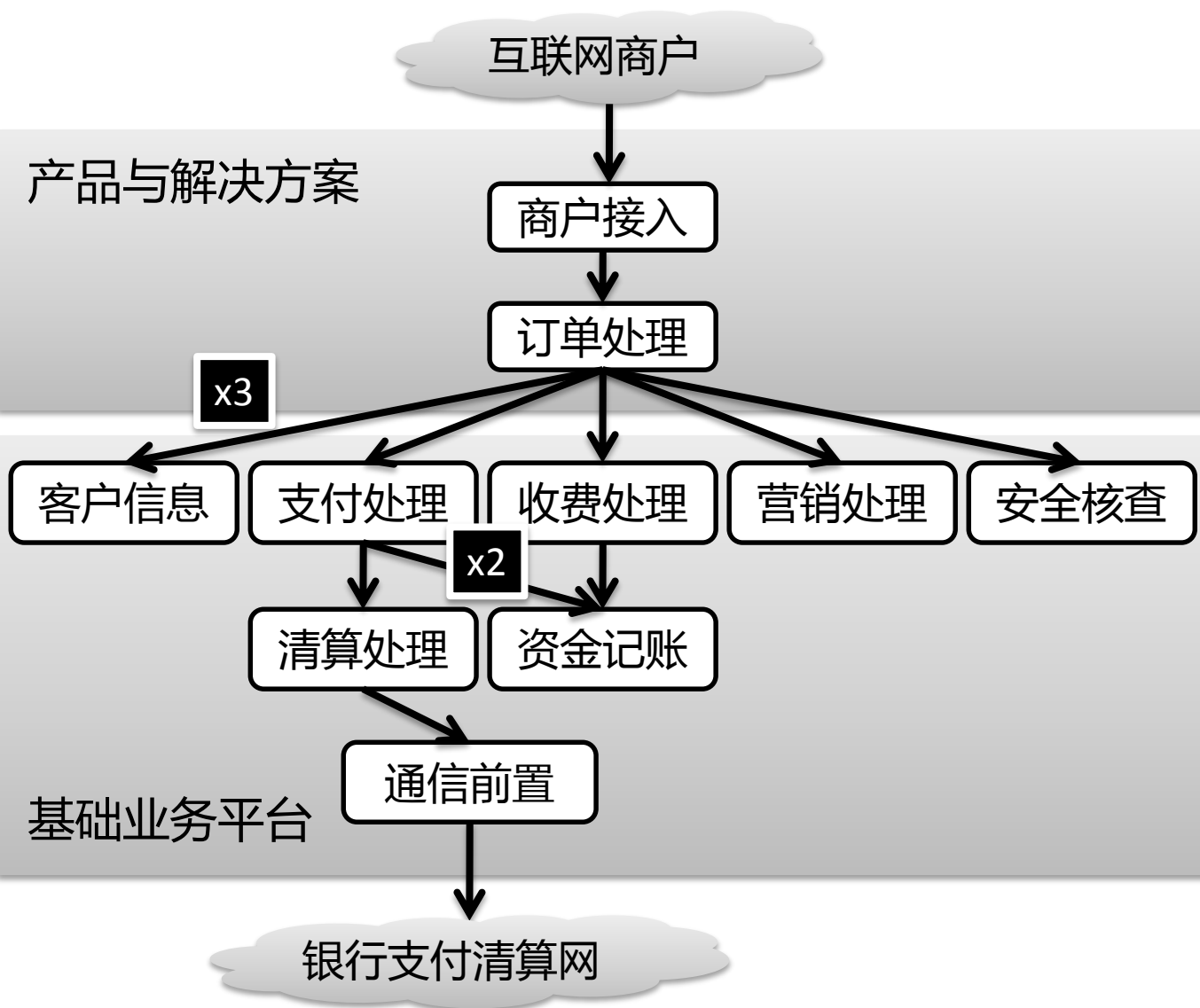
通过建设基础业务平台，达到快速构建与改进上层的产品与解决方案的目标。



# 交付前，你胸有成竹吗



- 性能
- 容量
- 健壮





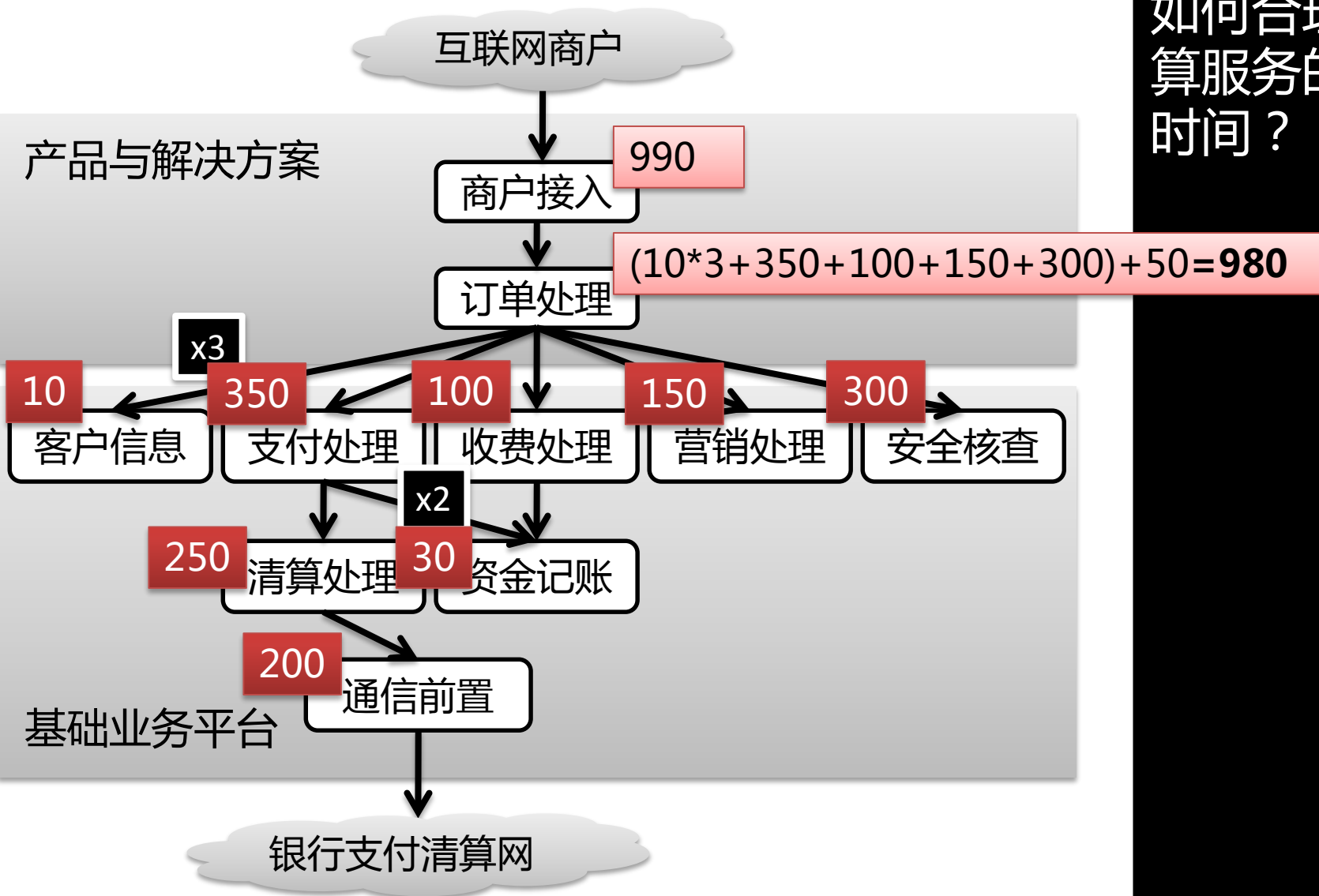
# 性能

## 针对性能的分析与优化



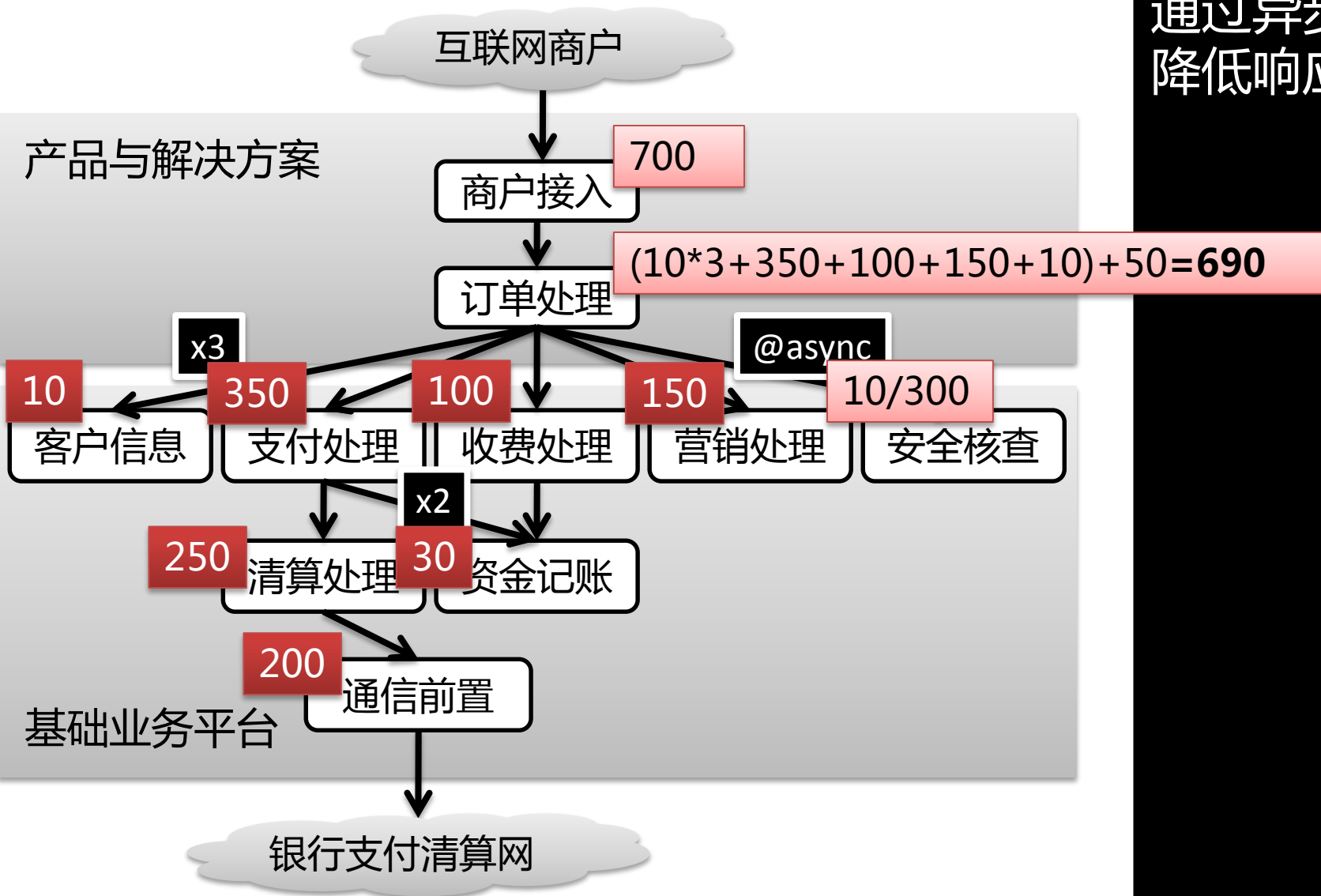
# 响应时间分析

如何合理地估算服务的响应时间？



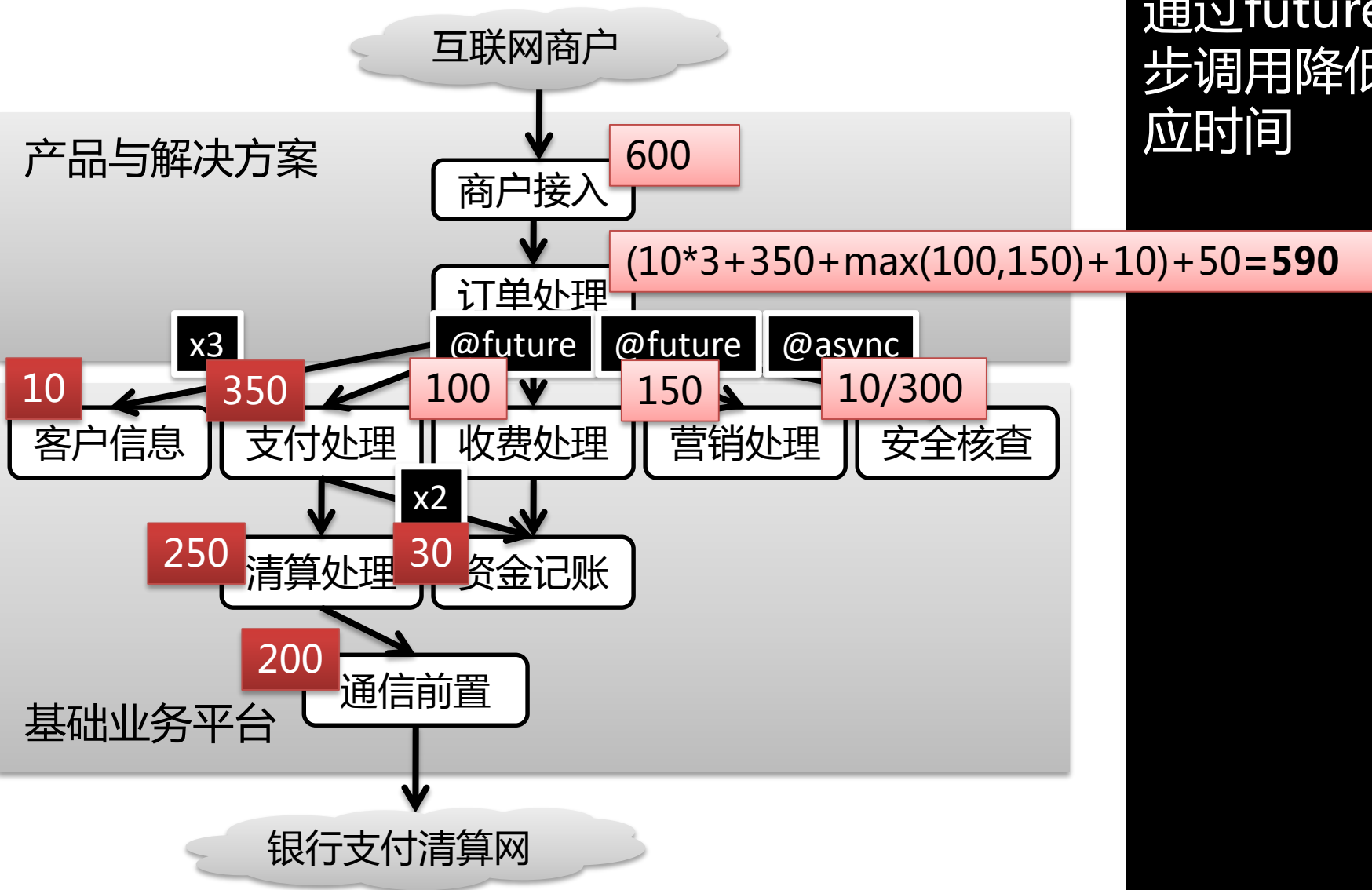
# 响应时间优化

通过异步调用  
降低响应时间

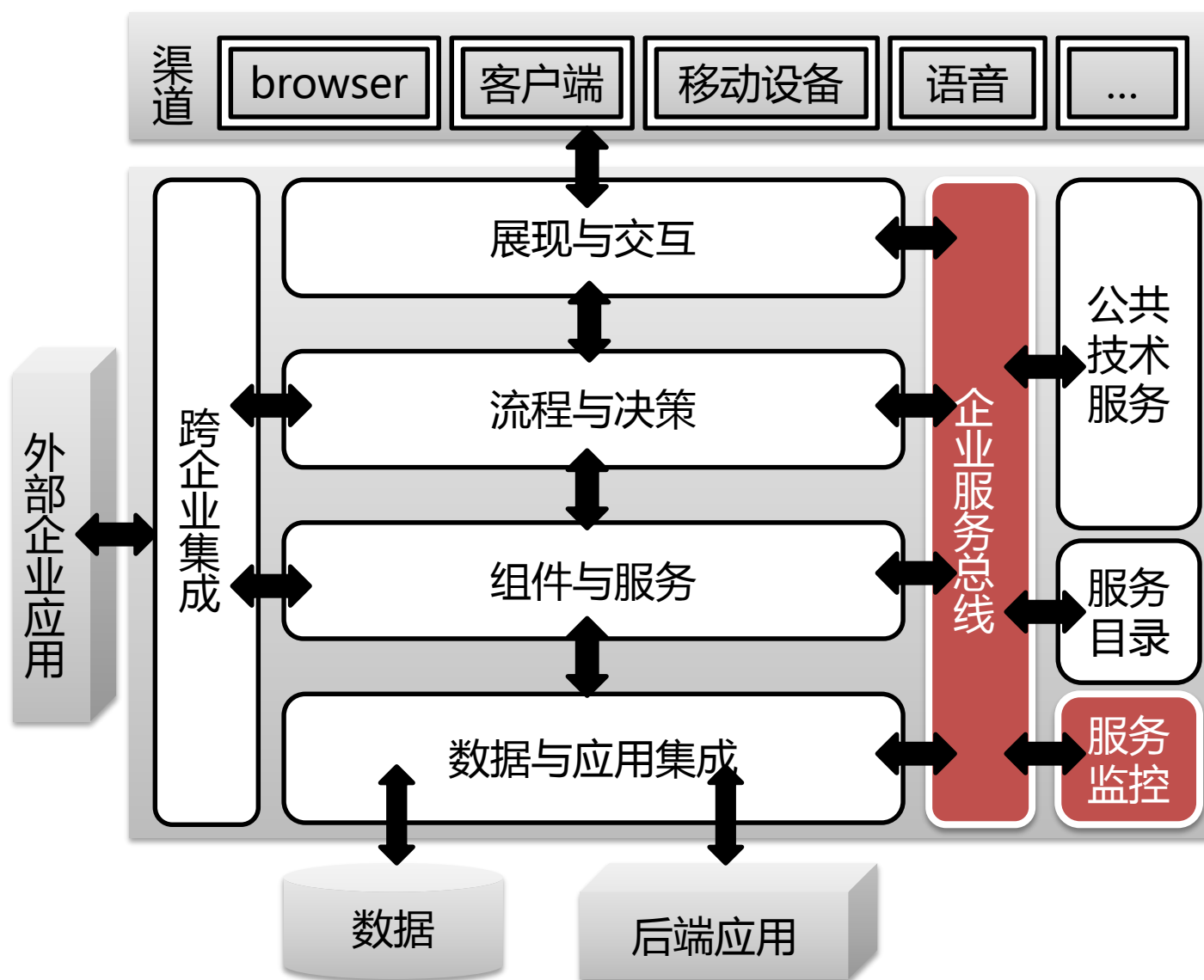


# 响应时间优化

通过future异步调用降低响应时间



# 关于性能的基础设施支持



➤ 知晓所有服务的响应时间  
数据: 服务监控

➤ 支持各种异步服务使用: 服务通信总线

# 小结

- 在设计阶段就必须估算与优化性能
- 准确估算性能依赖于真实的监控数据，尤其是业务平台的性能监控数据
- 灵活的服务通信设施使提升性能成为可能

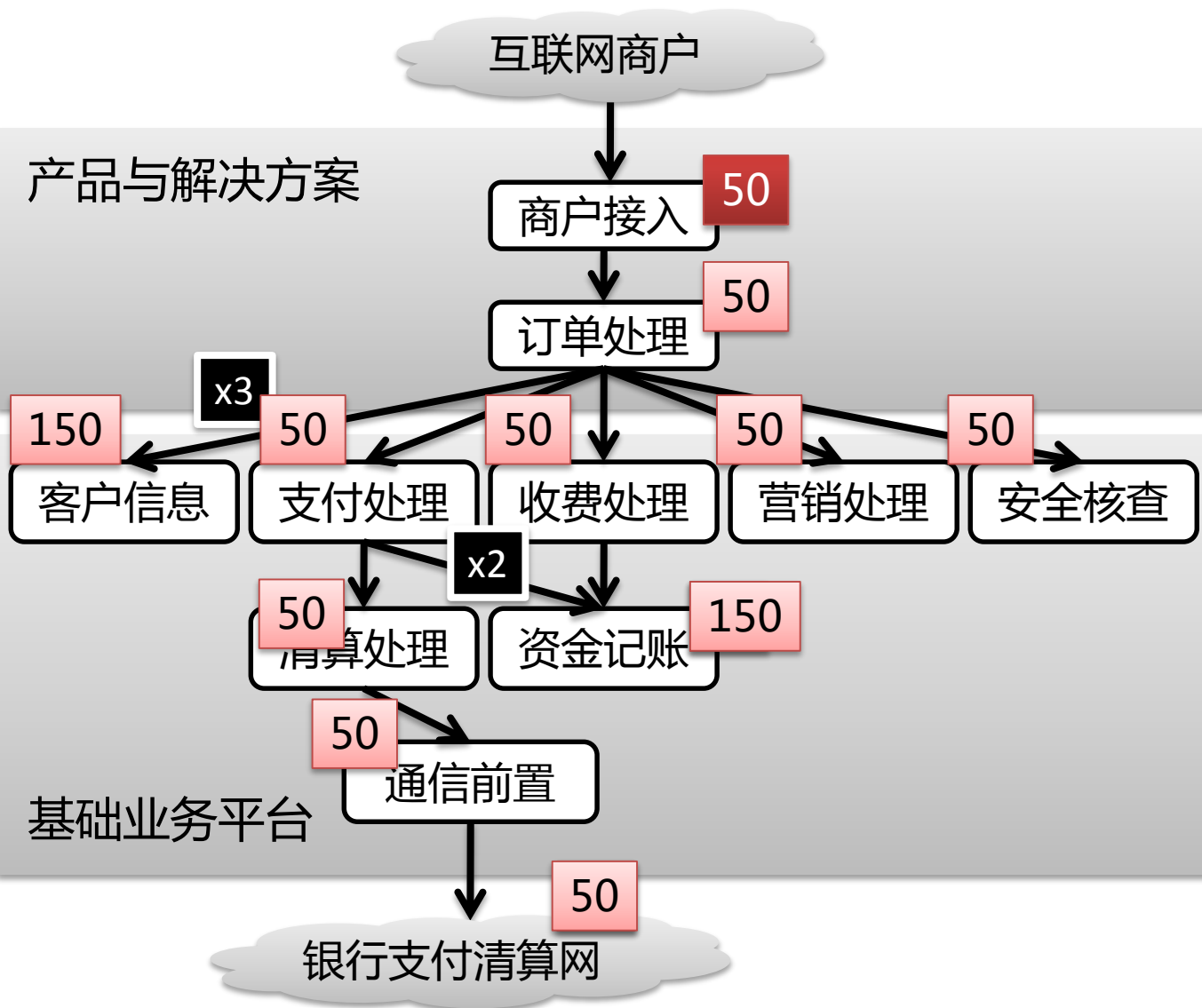


# 容量

## 针对容量的分析与优化

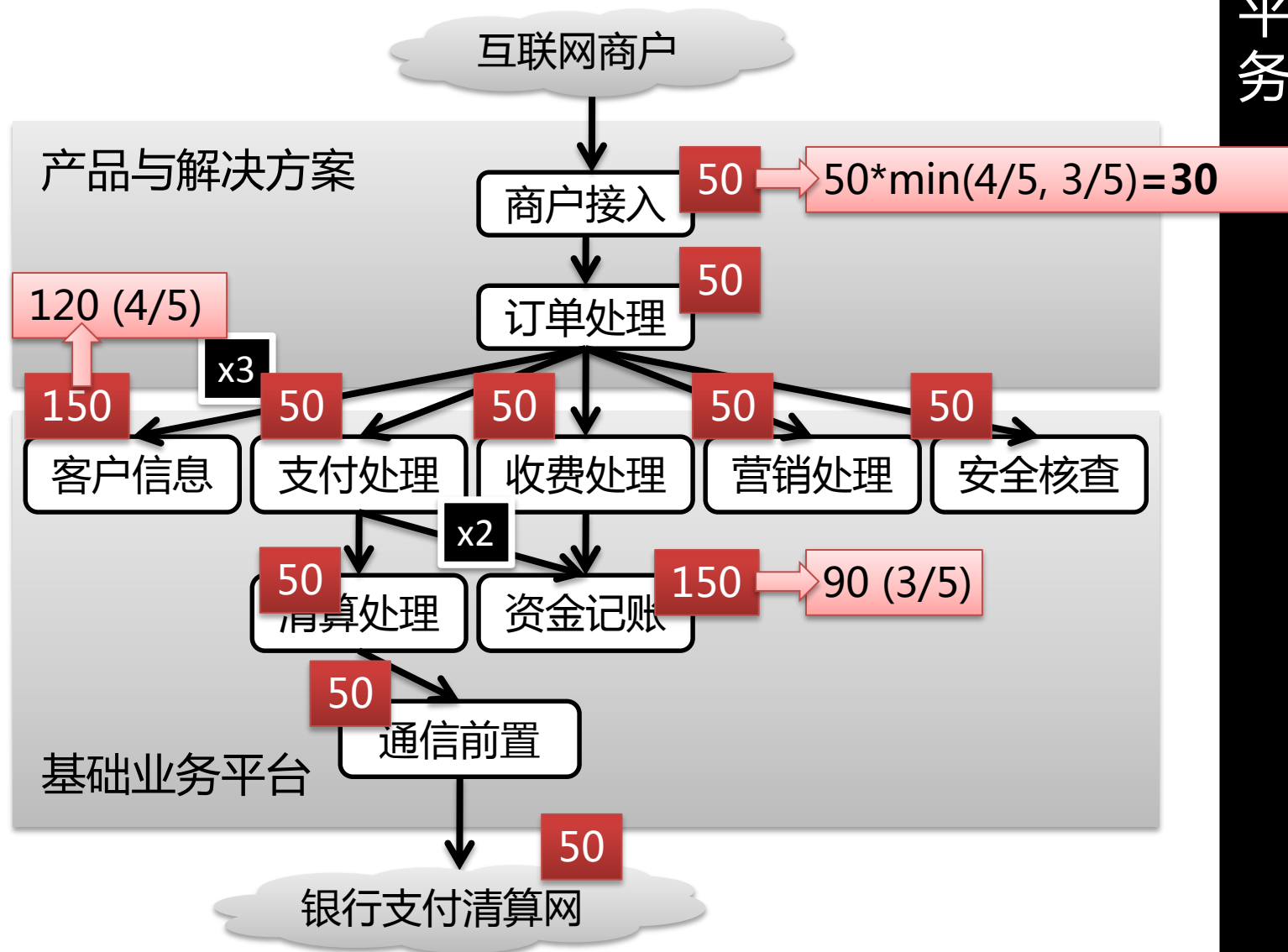
# 吞吐量分析

如何合理地估算新业务上线对容量的需求？



# 吞吐量分析

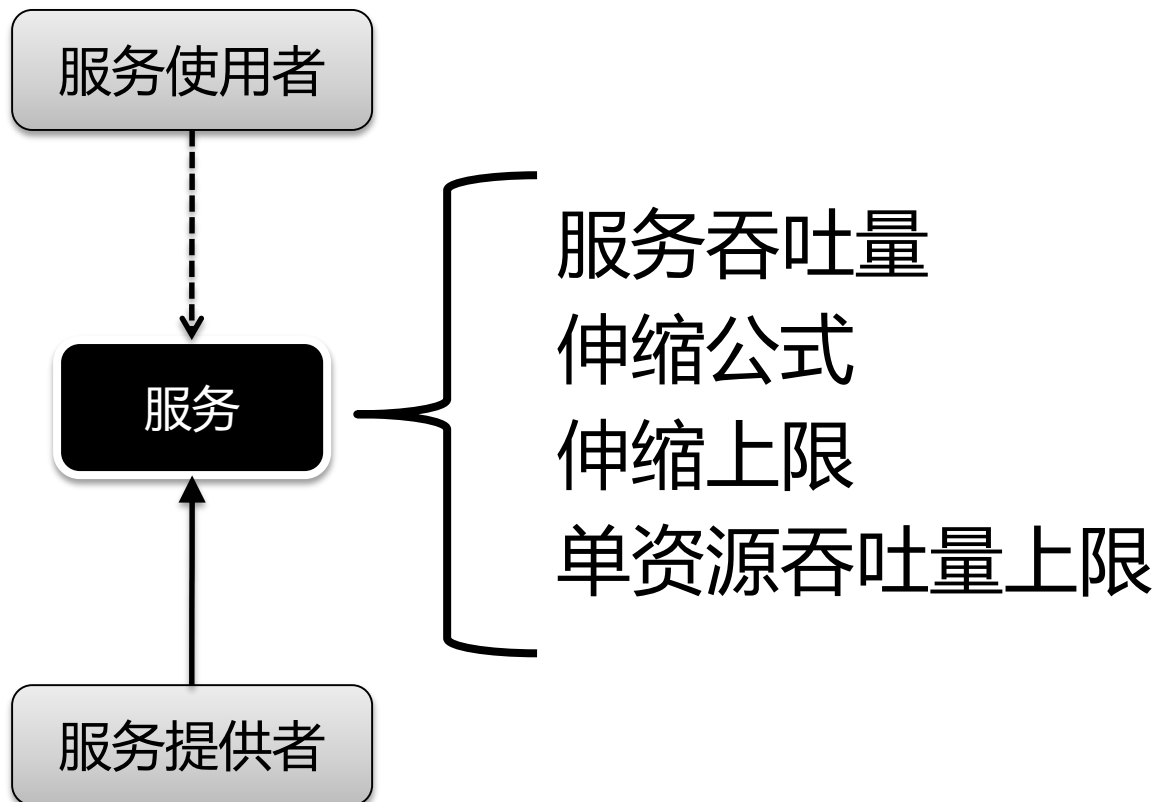
平台容量对业务容量的约束





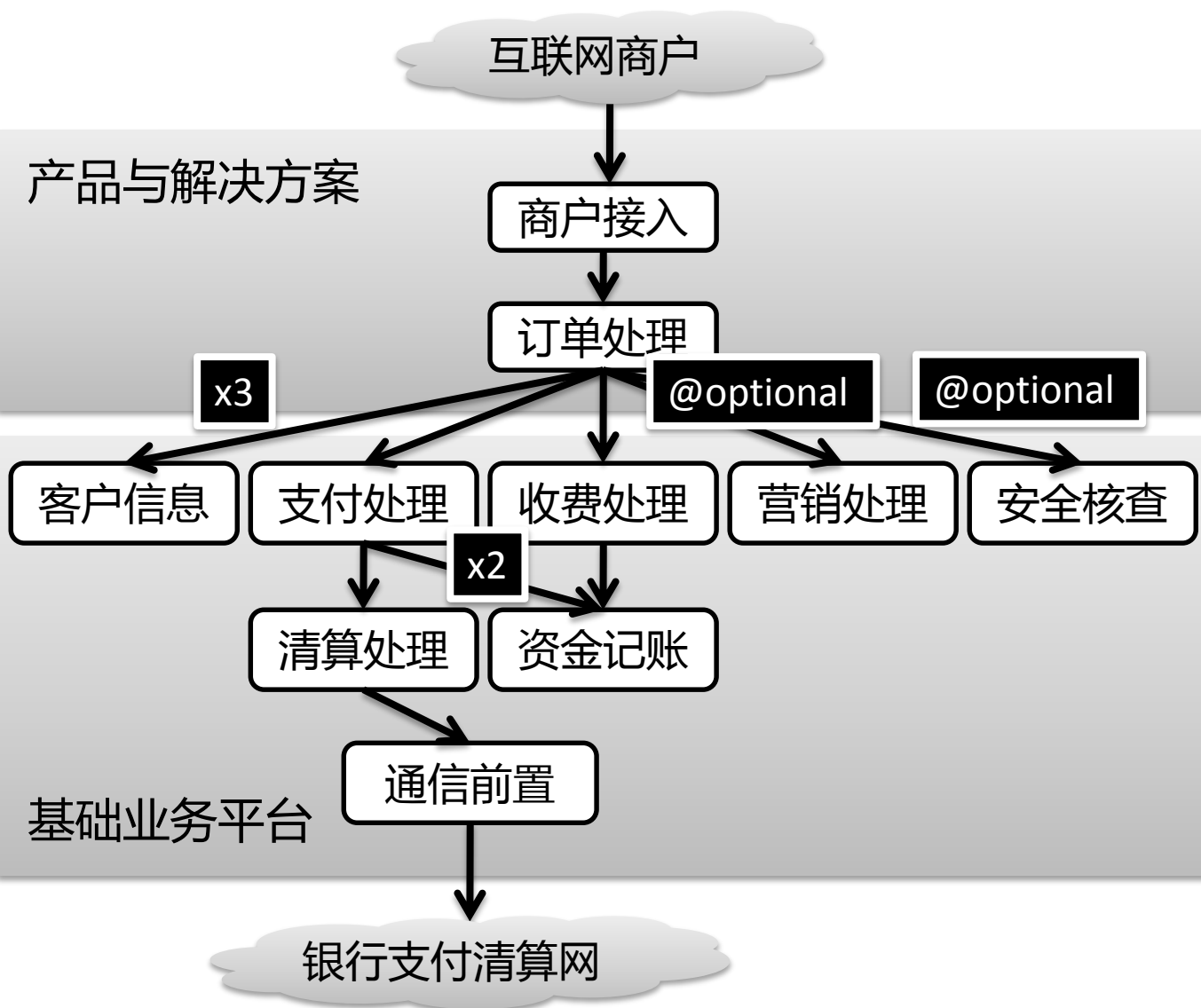
# 关键服务的吞吐量优化

- 充分扩容
- 平衡扩容
- 消除资源单点瓶颈



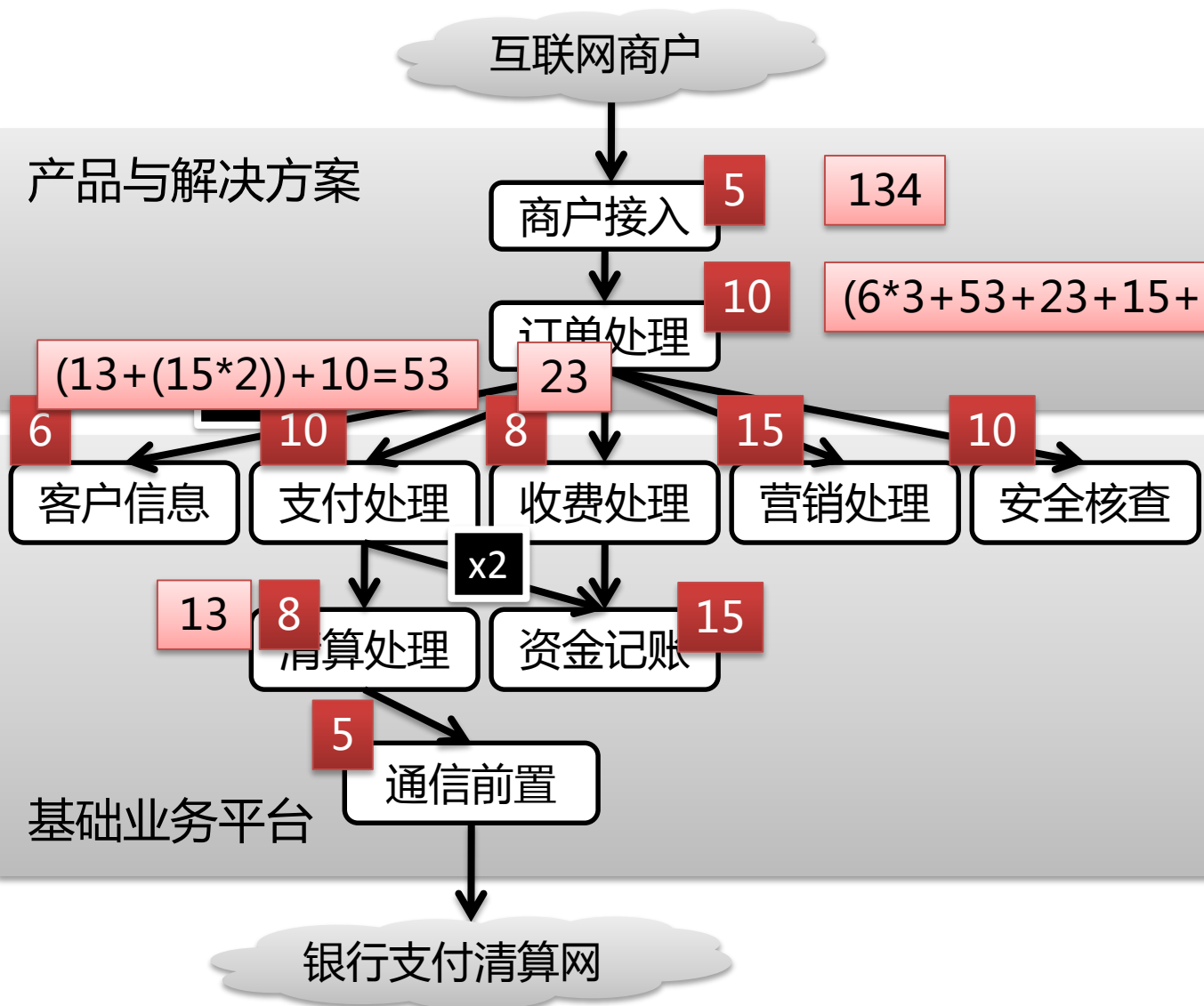
# 非关键服务的吞吐量优化

非关键的业务服务的容量允许短路，提供降级服务。

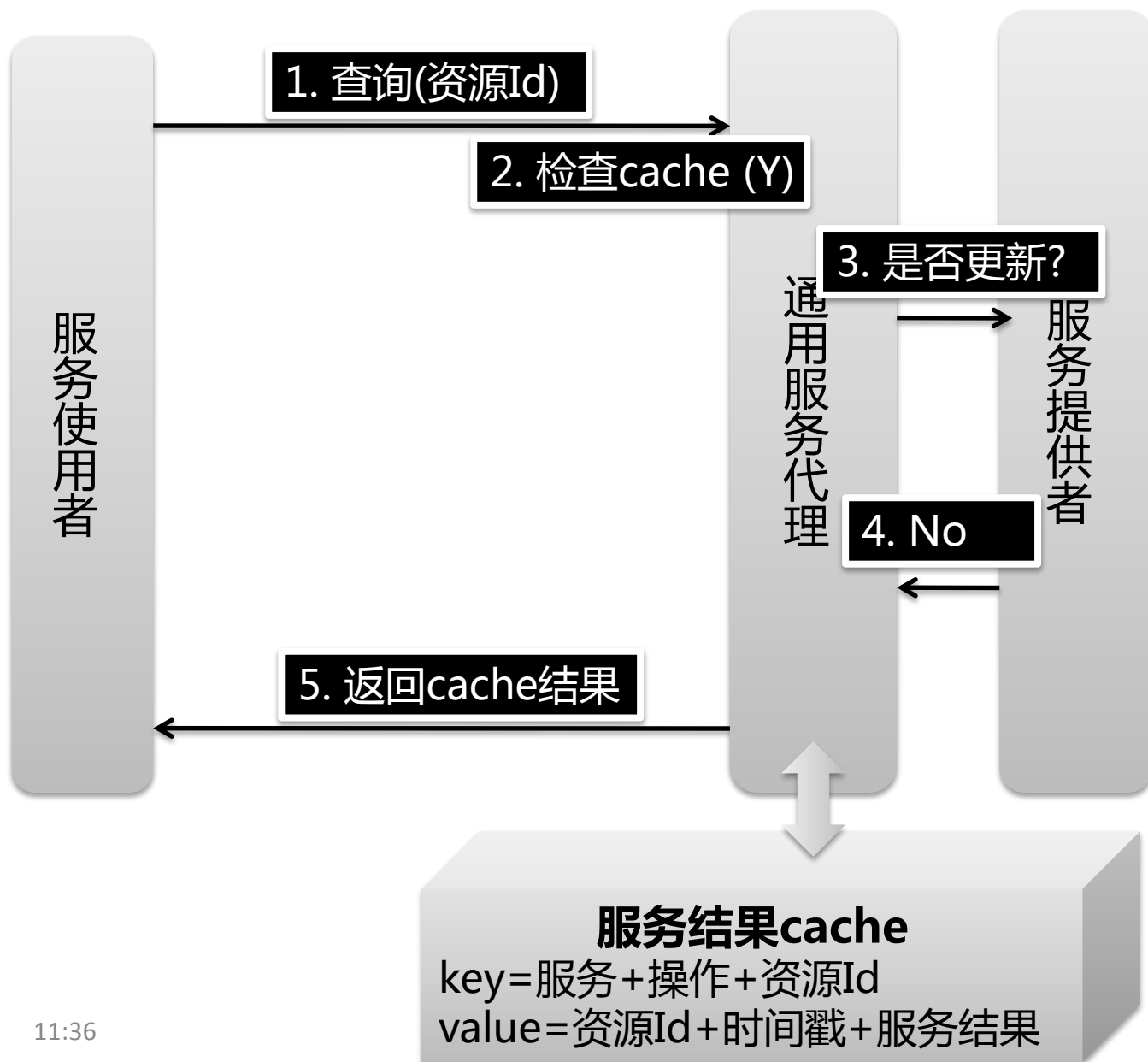


# 资源使用分析

估算新业务对  
关键资源的使用 (以SQL执行  
次数为例)



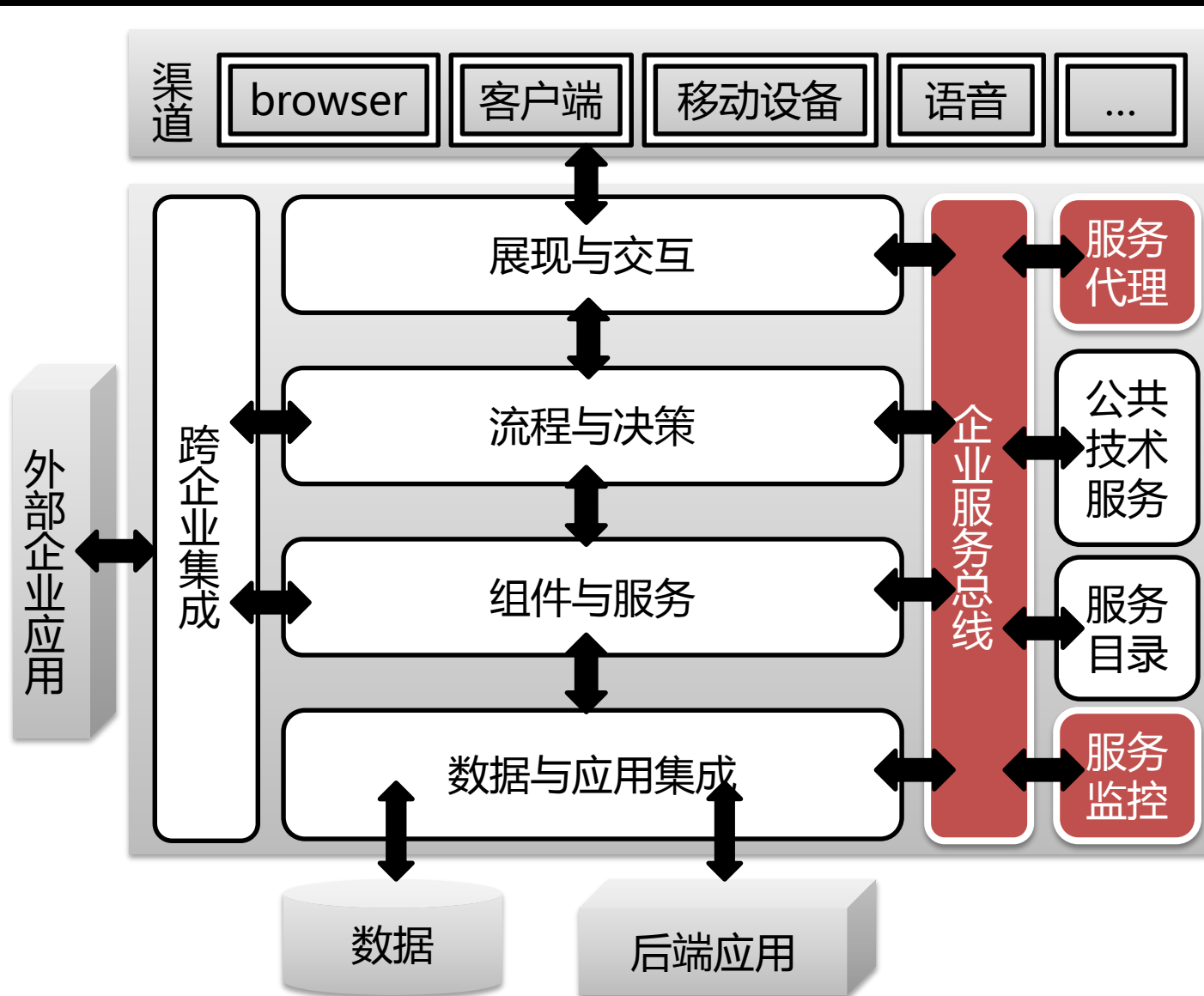
# 资源使用优化



➤通用服务代理缓存服务结果。

➤服务提供者支持检查资源更新时间戳。

# 关于容量的基础设施支持



➤ 知晓所有服务的吞吐量与资源使用: 服务监控

➤ 支持 optional 服务使用: 服务通信总线

➤ 服务结果 cache: 服务代理

# 小结

- 在设计阶段就必须估算与优化容量
- 充分、平衡对业务平台进行扩容，既有前瞻性又控制成本
- 针对“热点”进行优化
- 准确估算容量依赖于真实的监控数据
- 区别业务的等级

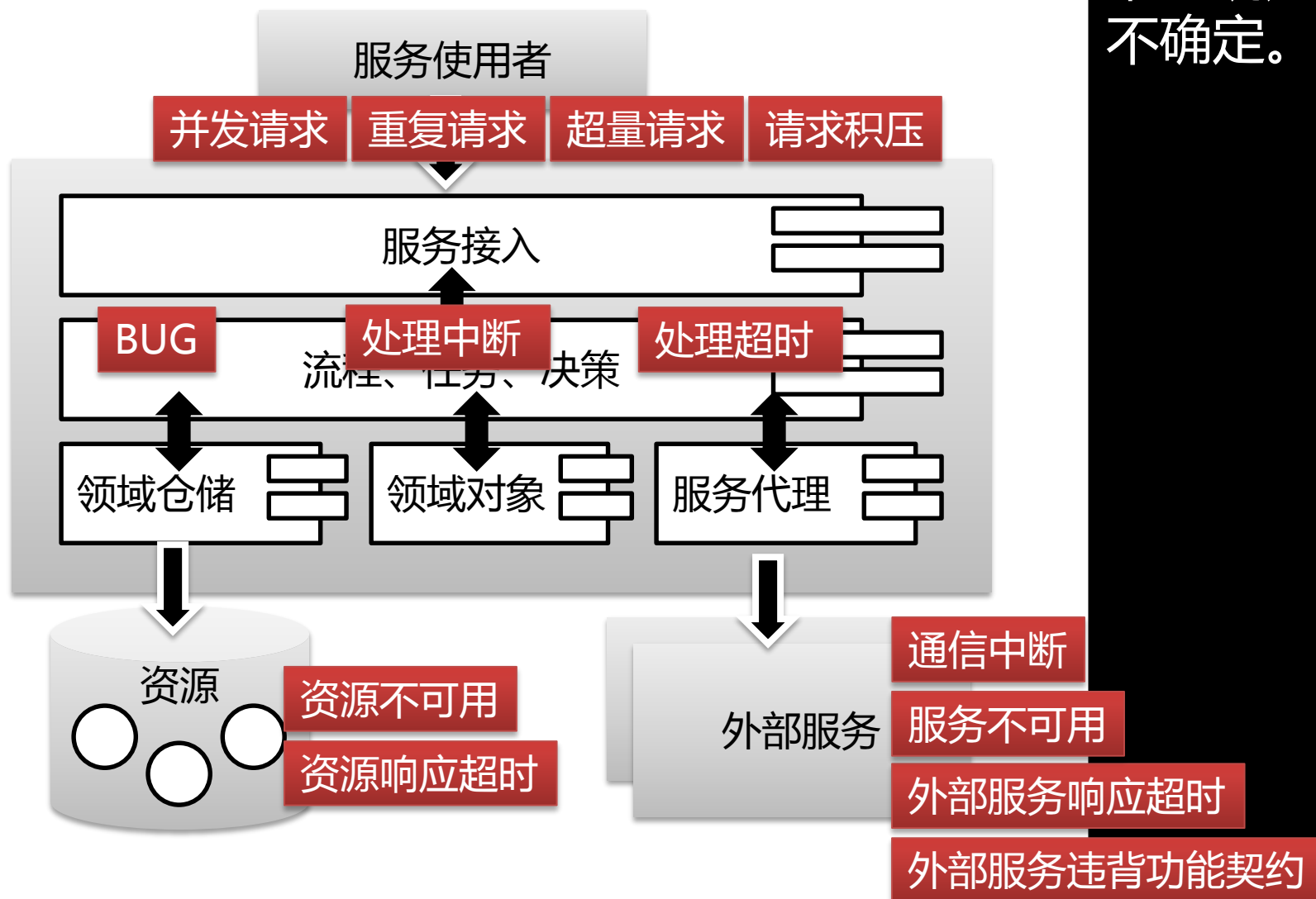


健壮

在不确定的世界中交付确定的服务

# 单个服务的故障条件

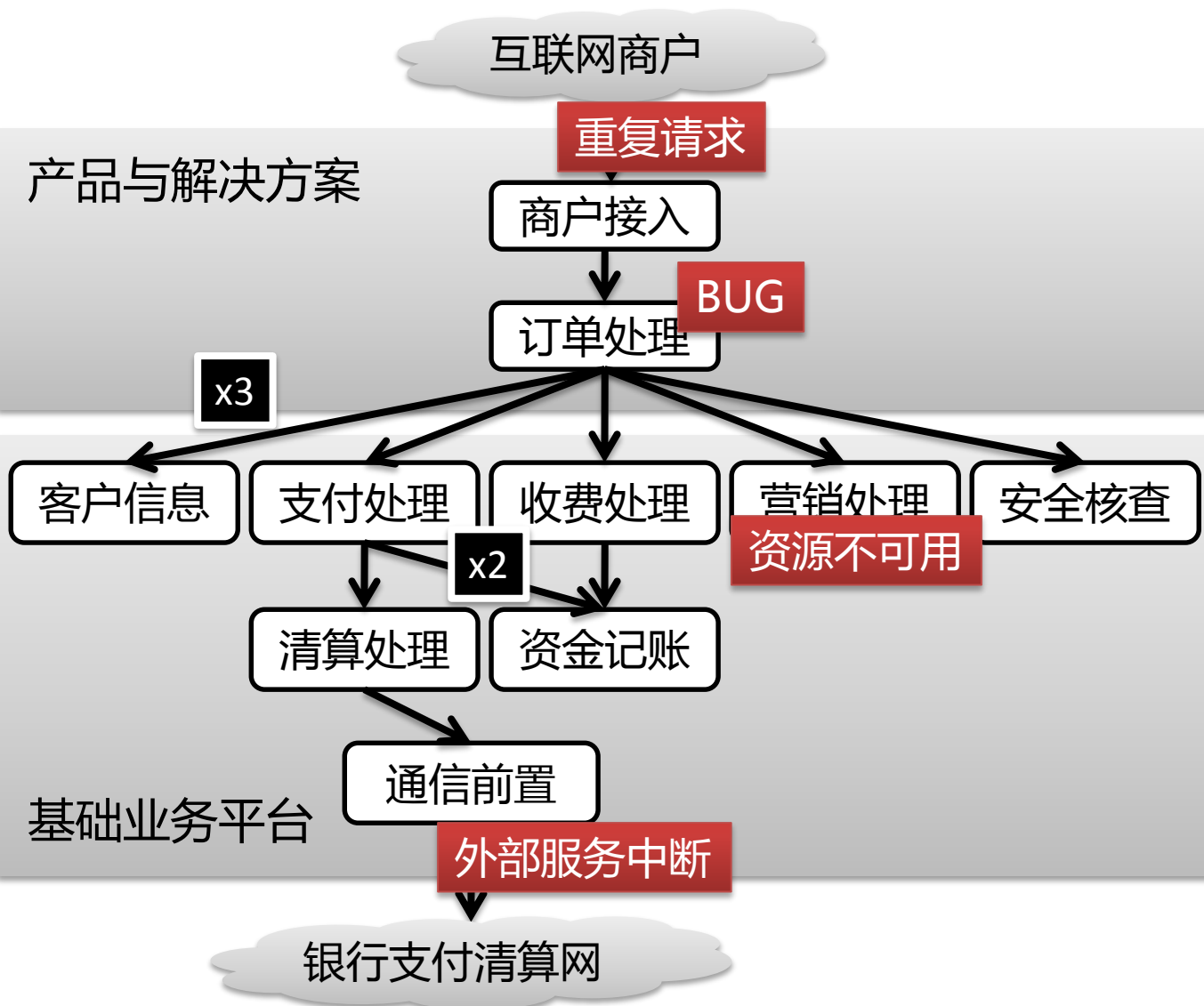
唯一确定的是  
不确定。





# 故障空间组合爆炸

处处都有多种故障可能，可能穷尽其组合吗？

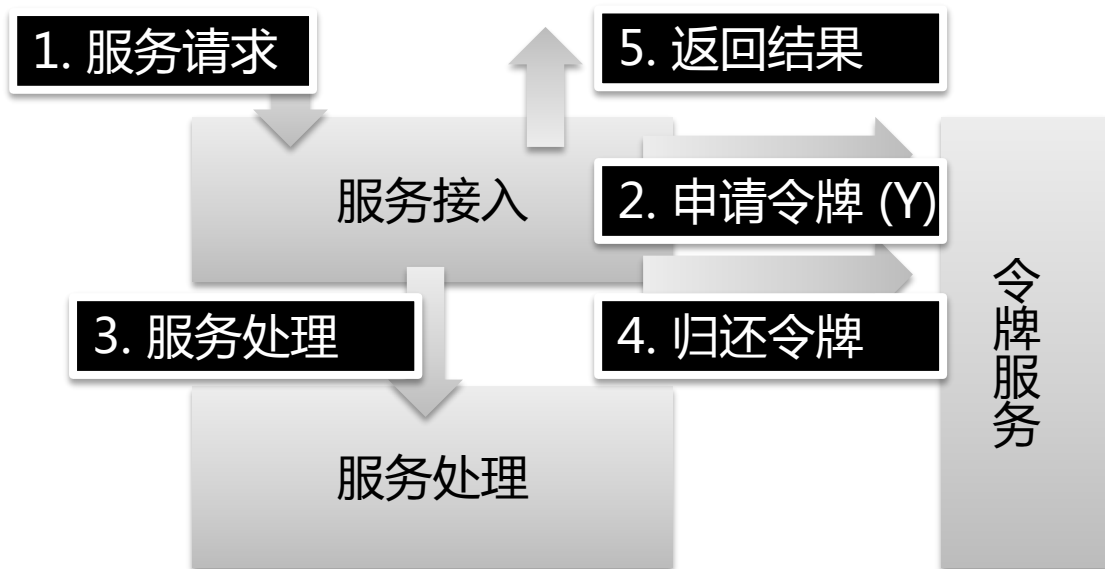


# 应对方式

故障条件	应对方式
超量请求	配额控制
重复请求	幂等控制
并发请求	并发控制
请求积压	请求丢弃
服务/资源响应超时	时间控制
可恢复通信故障	合理重试
处理中断	事务/分布事务
BUG	自检

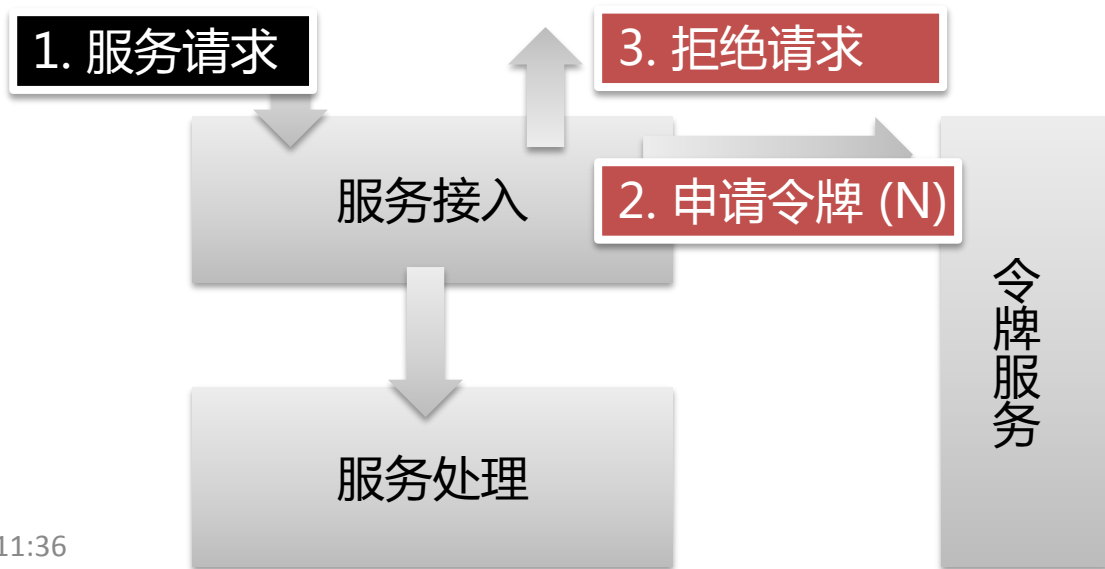
- 避免发生
- 降低概率
- 控制影响
- 快速恢复

# 局部配额控制

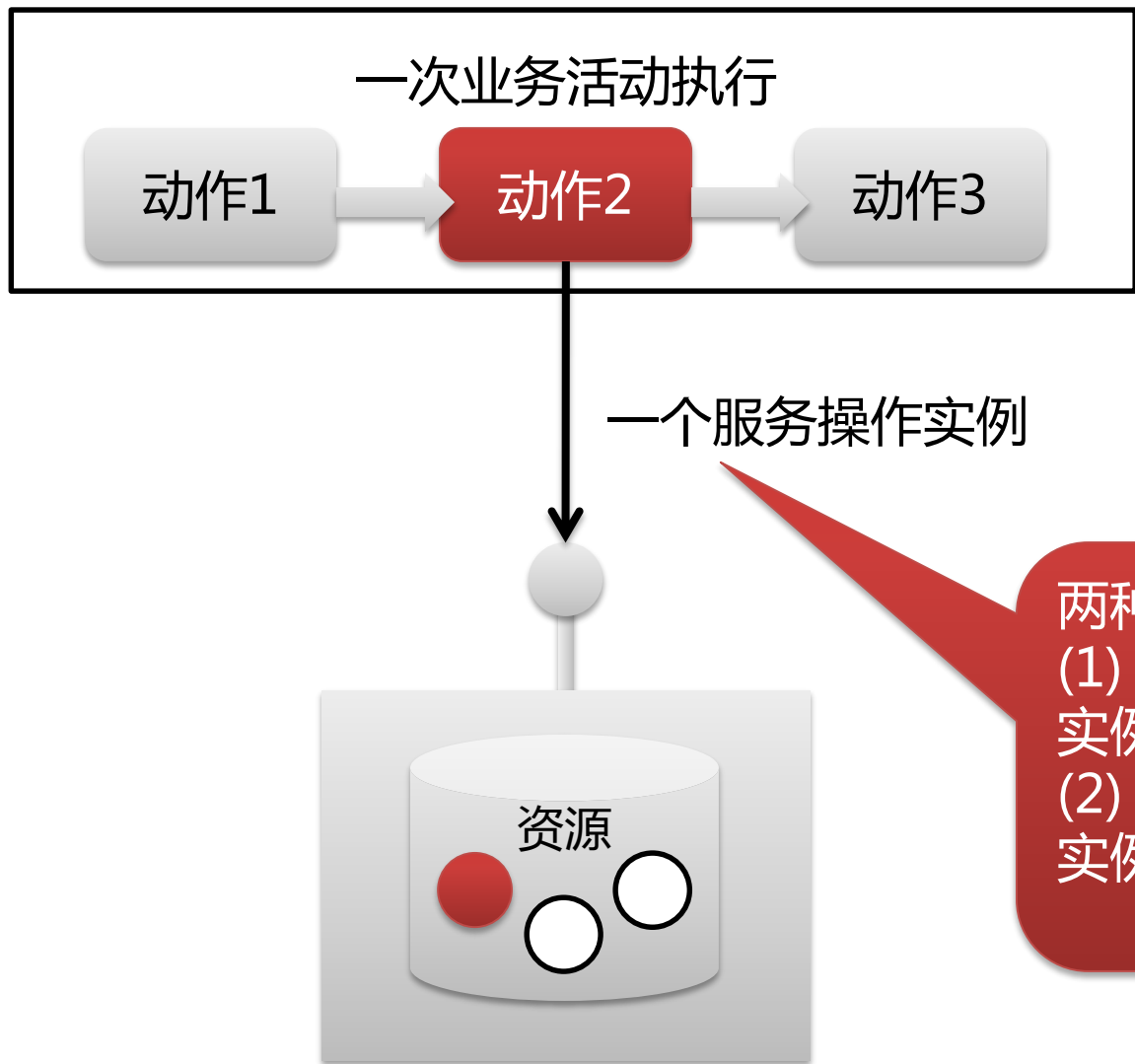


➤一种简单的基于令牌的配额控制方法。

➤令牌服务维持本地各个服务的可用令牌数。



# 幂等服务



同一个服务操作实例最多只允许执行一次。

两种操作实例标识法:

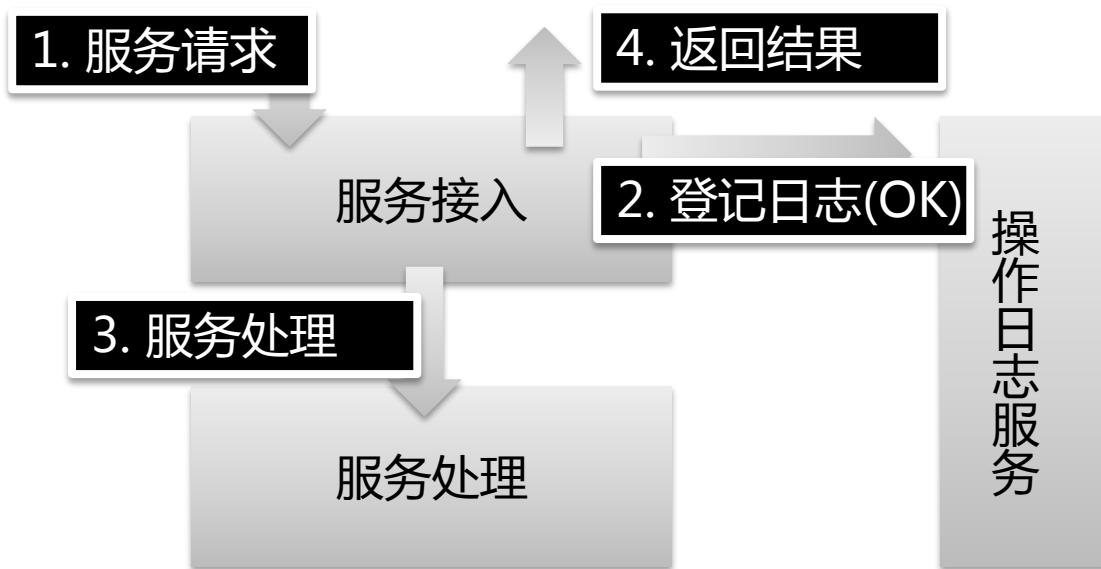
(1) 基于业务活动

实例Id = 业务活动Id + 操作序号

(2) 基于资源

实例Id = 资源Id + 操作类型

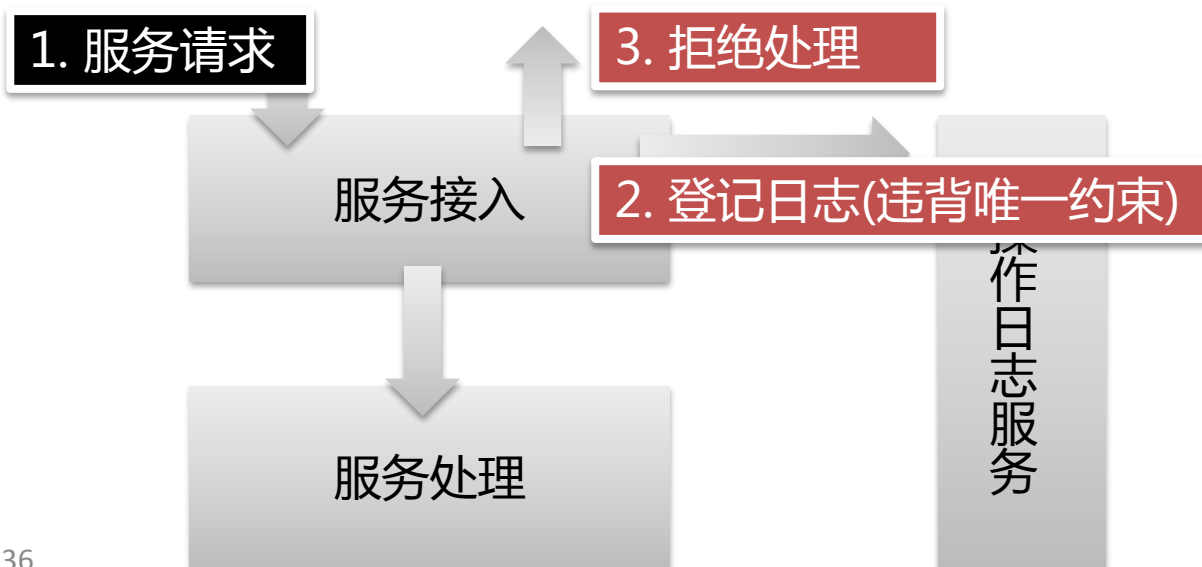
# 幂等控制



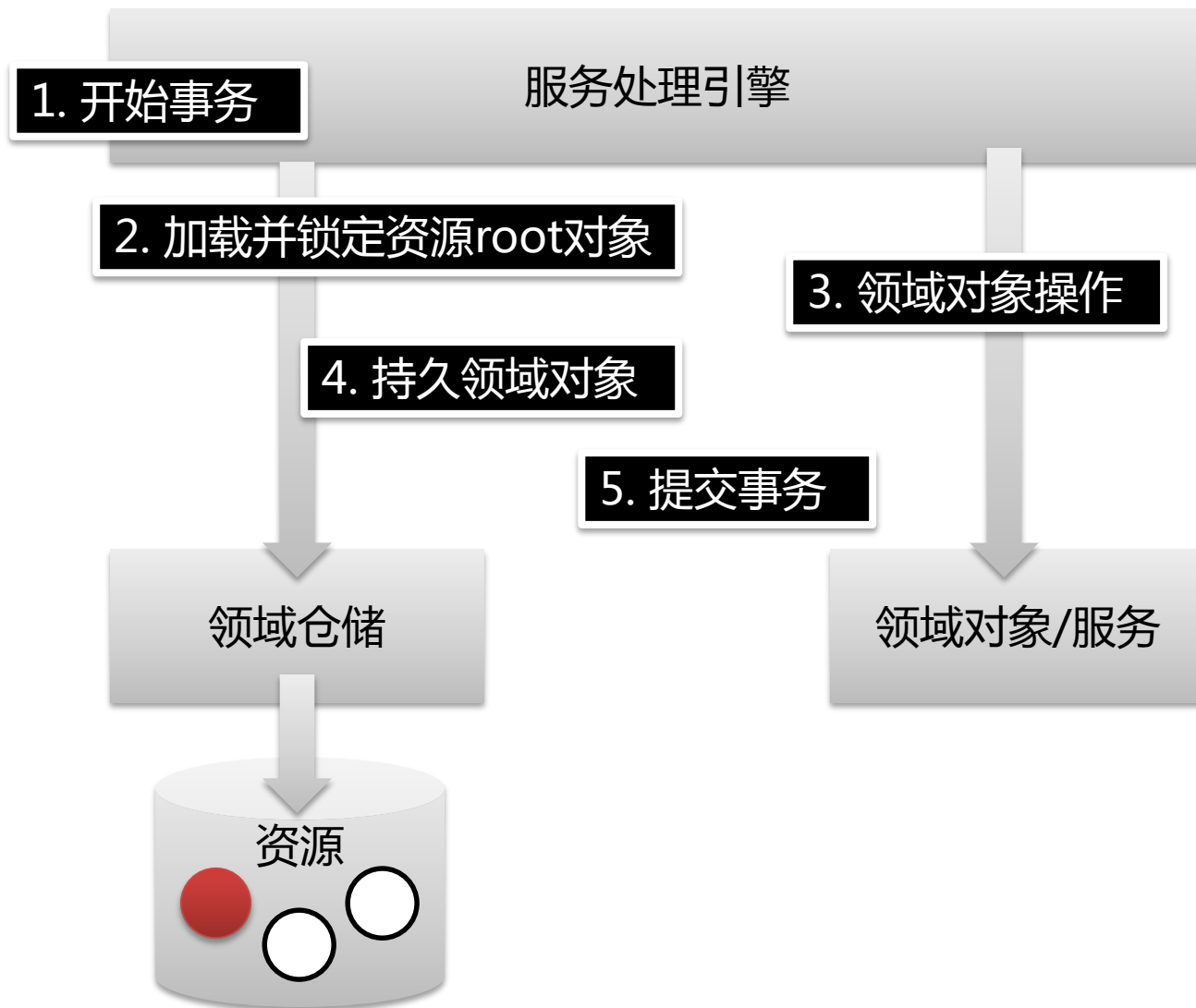
➤一种简单的基于操作日志的幂等控制方法。

➤操作实例Id上建立唯一性约束。

➤操作日志不仅用于幂等控制，还可用于操作审计等用途。



# 基于资源的并发控制 (悲观)

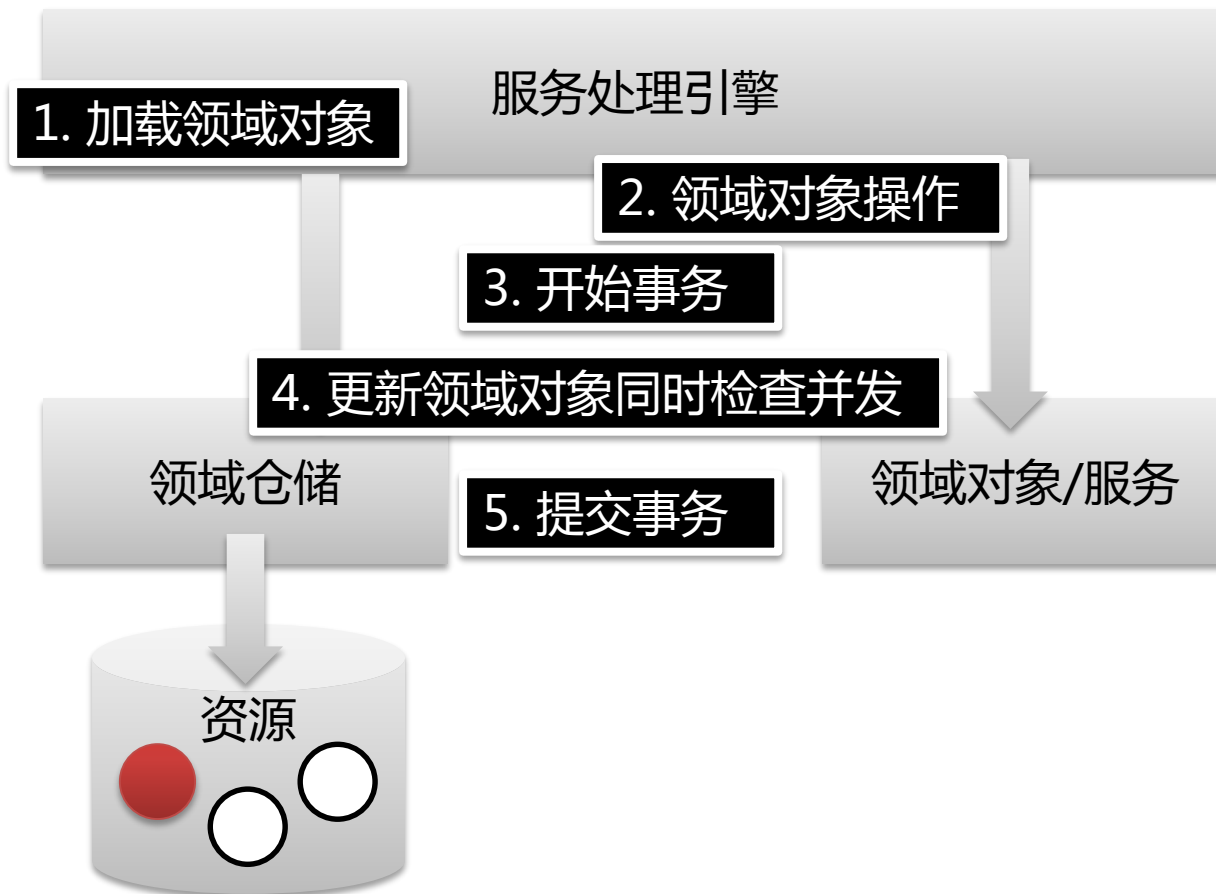


➤经典的资源并发控制方式。

➤事务长度限制了系统伸缩能力。

➤不适用热点资源。

# 基于资源的并发控制(乐观)

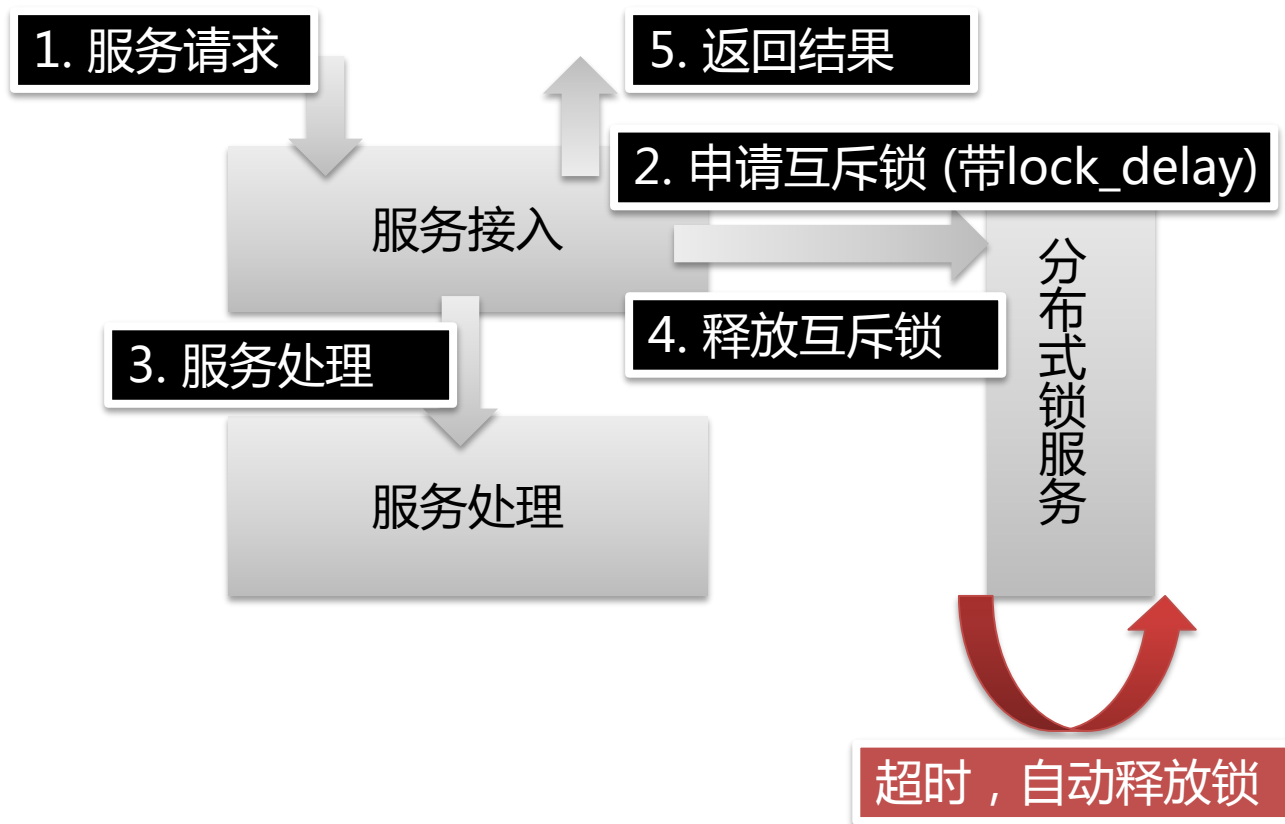


➤同样经典的资源并发控制方式。

➤事务长度短，提高系统伸缩性。

➤同样不适用于热点资源。

# 基于分布式锁服务的并发控制



没有可加锁的资源，怎么办？

➤以操作实例Id作为锁标识。

➤每个锁都有生命周期(lock\_delay)

➤过期锁被自我释放。

➤服务处理时间不应超过lock\_delay。

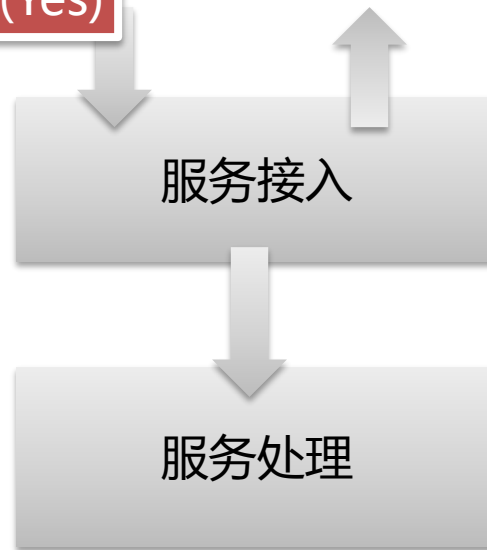


# 请求丢弃

1. 从队列中取出服务请求

2. 请求已过期？(Yes)

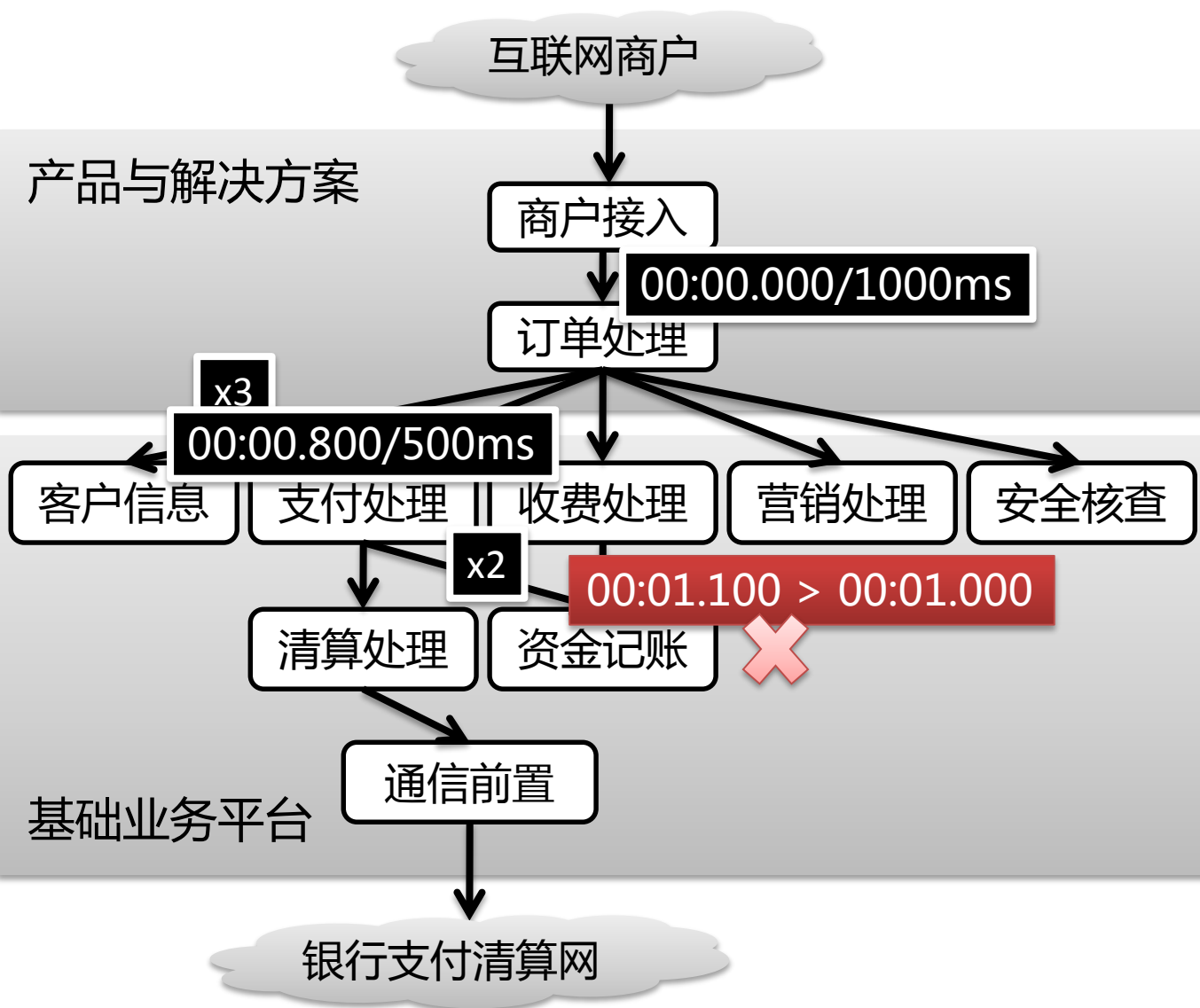
3. 丢弃请求



➤服务请求包含处理期限  
(= 请求发出时间 + 客户端超时设置)

➤已过期的请求直接丢弃，腾出宝贵服务处理资源。

# 时间控制



➤ 延误的处理比不处理更糟糕。

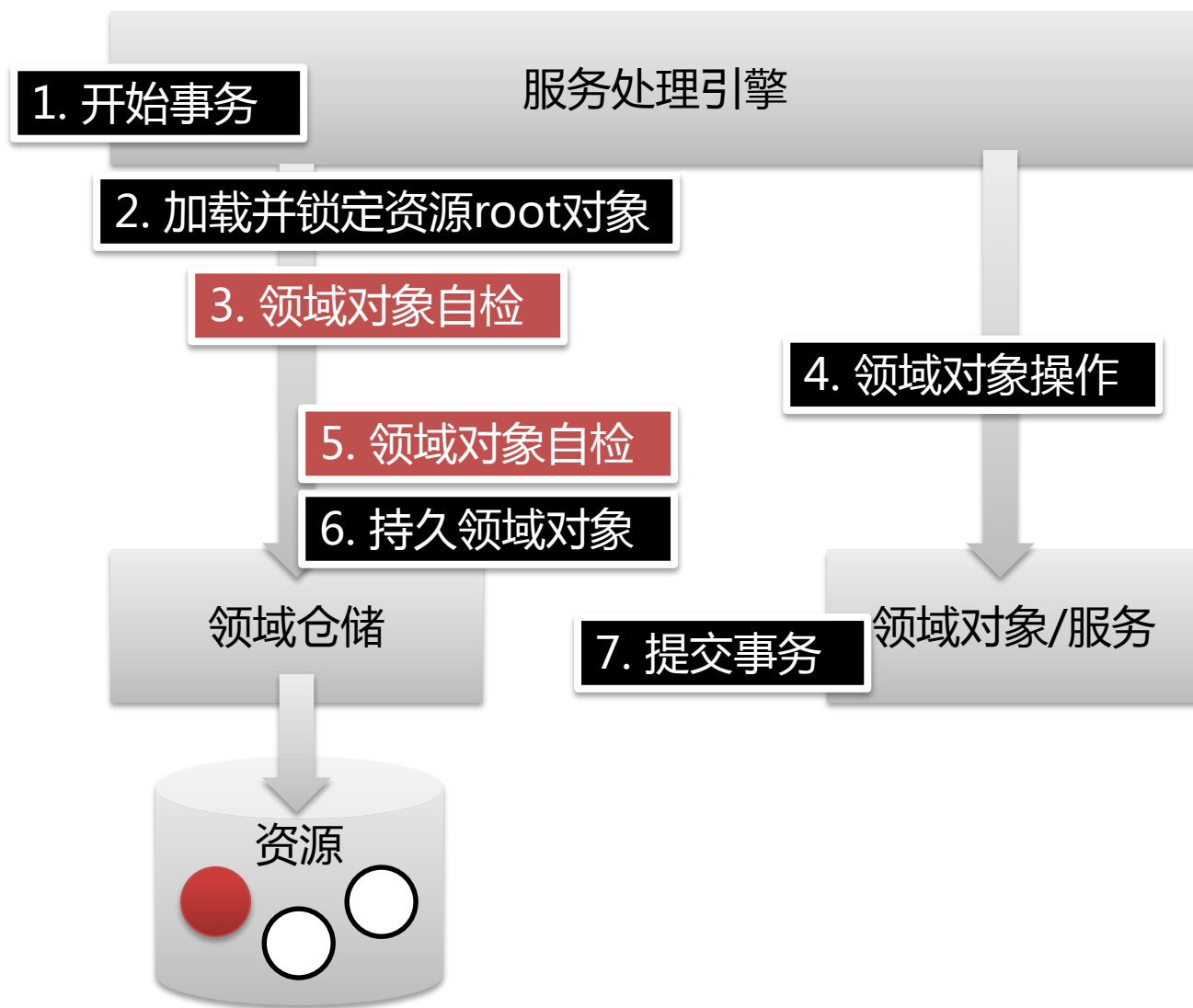
➤ 一个业务活动是否要继续处理，取决于整体期限。

➤ 整体期限在统一业务活动上下文中传输。

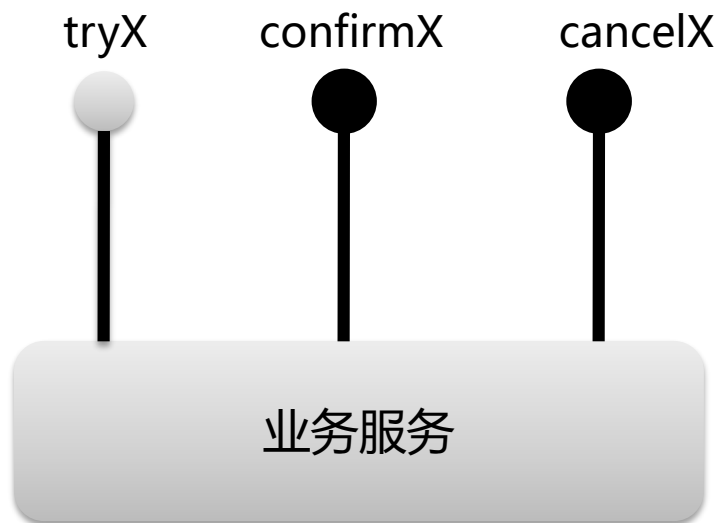
➤ 考虑服务器时间差异补偿。

# 领域自检

- 不变式
- 状态变迁



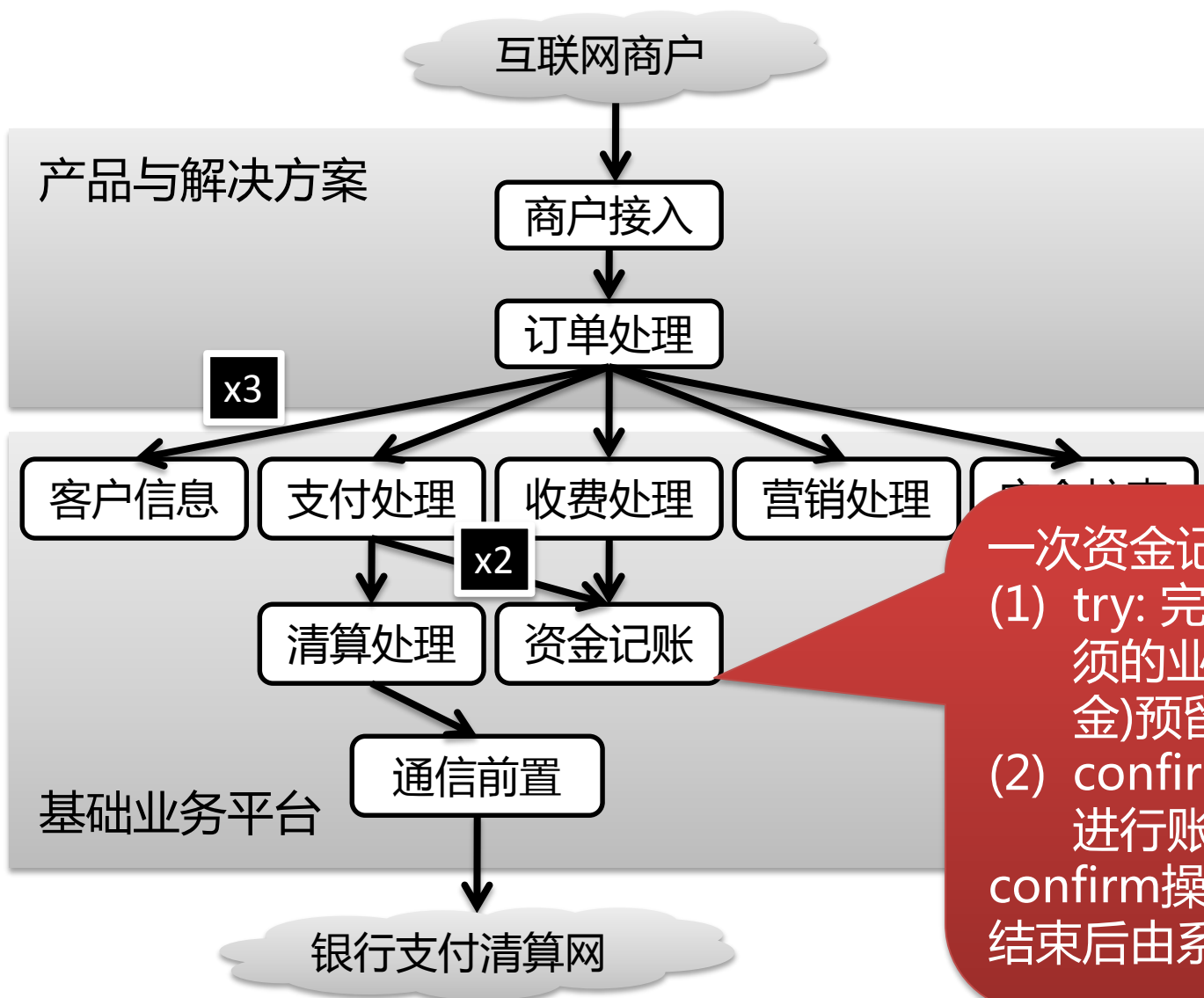
# 分布事务 (TCC模式)



- Try: 尝试执行业务
- Confirm: 确认执行业务
- Cancel: 取消执行业务

# 分布事务 (TCC模式)

基于TCC模式的分布事务执行过程示例。

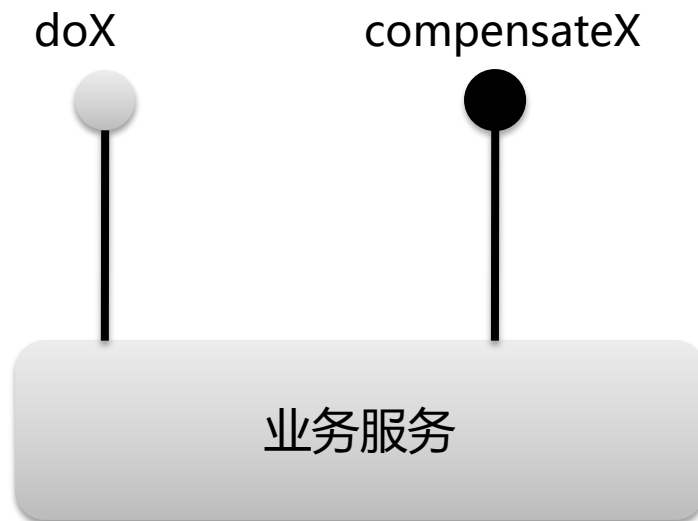


一次资金记账操作分为两步：  
(1) try: 完成所有账务处理必须的业务检查与资源(资金)预留。  
(2) confirm: 释放资源并真正进行账务处理。  
confirm操作在整个业务活动结束后由系统自动完成。

# 分布事务 (补偿模式)

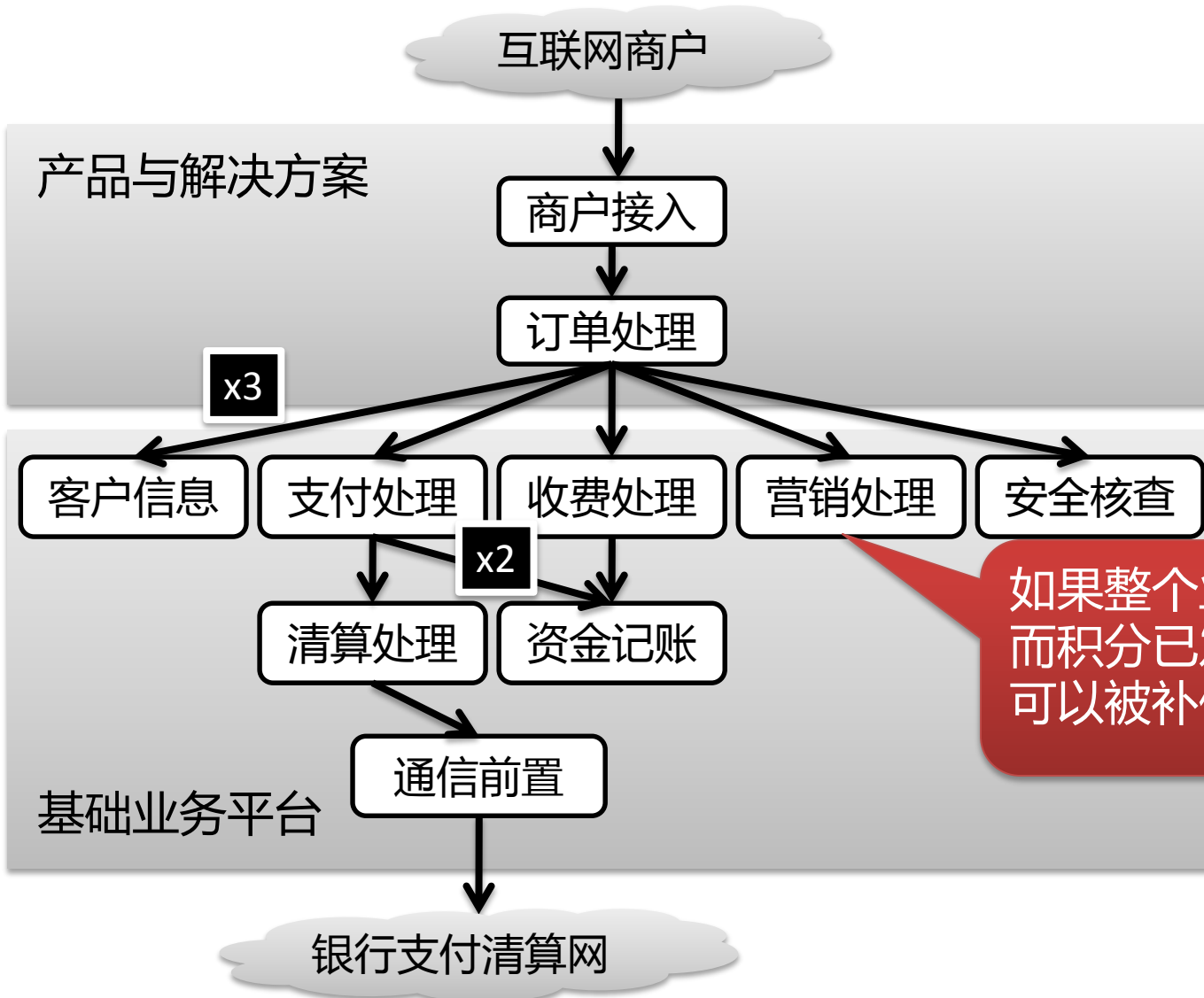
➤ do: 真正执行业务

➤ compensate: 业务补偿

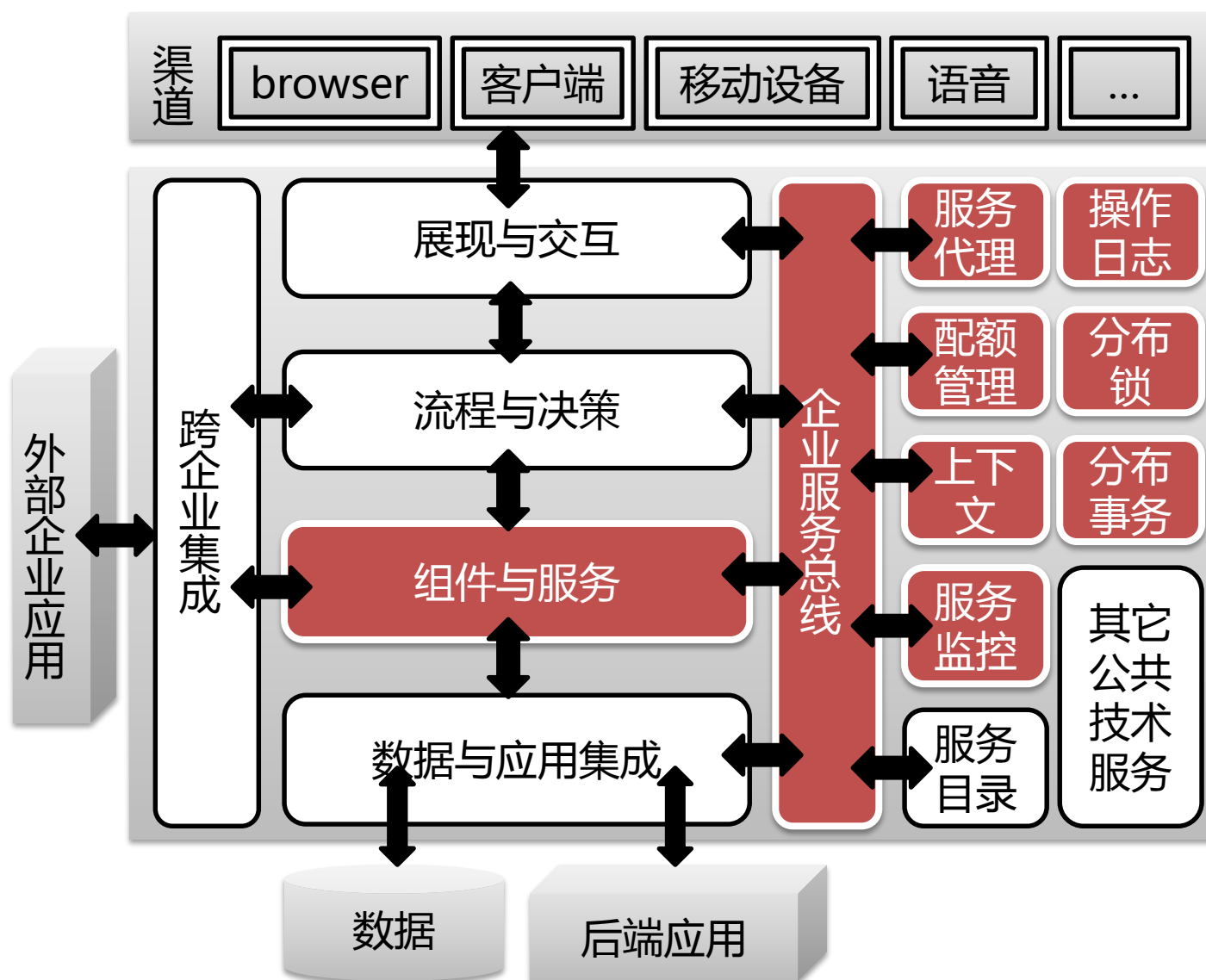


# 分布事务 (补偿模式)

一次补偿模式的业务执行过程。



# 关于健壮的基础设施支持



- 控制请求量: 配合管理服务
- 幂等控制: 操作日志服务
- 并发控制: 分布锁服务
- 时间控制: 上下文服务
- 事务控制: 分布事务服务



# 小结

- 生产环境是严酷的
- 设计对故障条件免疫的服务
- 设计可靠、可恢复的业务活动
- 健壮的应用需要大量SOA基础设施的支持

