# 服务拆分的规范

微服务拆分之后，工程会比较的多，如果没有一定的规范，将会非常混乱，难以维护。  
  
首先人们经常问的一个问题是，服务拆分之后，原来都在一个进程里面的函数调用，现在变成了A调用B调用C调用D调用E，会不会因为调用链路过长而使得相应变慢呢？

## 服务拆分的规范一：服务拆分最多三层，两次调用

服务拆分是为了横向扩展，因而应该横向拆分，而非纵向拆成一串的。也即应该将商品和订单拆分，而非下单的十个步骤拆分，然后一个调用一个。  
  
纵向的拆分最多三层：

基础服务层：用于屏蔽数据库，缓存层，提供原子的对象查询接口，有这一层，为了数据层做一定改变的时候，例如分库分表，数据库扩容，缓存替换等，对于上层透明，上层仅仅调用这一层的接口，不直接访问数据库和缓存。

组合服务层：这一层调用基础服务层，完成较为复杂的业务逻辑，实现分布式事务也多在这一层

Controller层：接口层，调用组合服务层对外

## 服务拆分的规范二：仅仅单向调用，严禁循环调用

微服务拆分后，服务之间的依赖关系复杂，如果循环调用，升级的时候就很头疼，不知道应该先升级哪个，后升级哪个，难以维护。  
  
因而层次之间的调用规定如下：

基础服务层主要做数据库的操作和一些简单的业务逻辑，不允许调用其他任何服务。

组合服务层，可以调用基础服务层，完成复杂的业务逻辑，可以调用组合服务层，不允许循环调用，不允许调用Controller层服务

Controller层，可以调用组合业务层服务，不允许被其他服务调用

如果出现循环调用，例如A调用B，B也调用A，则分成Controller层和组合服务层两层，A调用B的下层，B调用A的下层。也可以使用消息队列，将同步调用，改为异步调用。

## 服务拆分的规范三：将串行调用改为并行调用，或者异步化

如果有的组合服务处理流程的确很长，需要调用多个外部服务，应该考虑如何通过消息队列，实现异步化和解耦。  
  
例如下单之后，要刷新缓存，要通知仓库等，这些都不需要再下单成功的时候就要做完，而是可以发一个消息给消息队列，异步通知其他服务。  
  
而且使用消息队列的好处是，你只要发送一个消息，无论下游依赖方有一个，还是有十个，都是一条消息搞定，只不过多几个下游监听消息即可。  
  
对于下单必须同时做完的，例如扣减库存和优惠券等，可以进行并行调用，这样处理时间会大大缩短，不是多次调用的时间之和，而是最长的那个系统调用时间。

## 服务拆分的规范四：接口应该实现幂等

微服务拆分之后，服务之间的调用当出现错误的时候，一定会重试，但是为了不要下两次单，支付两次，需要所有的接口实现幂等。  
  
幂等一般需要设计一个幂等表来实现，幂等表中的主键或者唯一键可以是transaction id，或者business id，可以通过这个id的唯一性标识一个唯一的操作。  
  
也有幂等操作使用状态机，当一个调用到来的时候，往往触发一个状态的变化，当下次调用到来的时候，发现已经不是这个状态，就说明上次已经调用过了。  
  
状态的变化需要是一个原子操作，也即并发调用的时候，只有一次可以执行。可以使用分布式锁，或者乐观锁CAS操作实现。

## 服务拆分的规范五：接口数据定义严禁内嵌，透传

微服务接口之间传递数据，往往通过数据结构，如果数据结构透传，从底层一直到上层使用同一个数据结构，或者上层的数据结构内嵌底层的数据结构，当数据结构中添加或者删除一个字段的时候，波及的面会非常大。  
  
因而接口数据定义，在每两个接口之间约定，严禁内嵌和透传，即便差不多，也应该重新定义，这样接口数据定义的改变，影响面仅仅在调用方和被调用方，当接口需要更新的时候，比较可控，也容易升级。

## 服务拆分的规范六：规范化工程名

微服务拆分后，工程名非常多，开发人员，开发团队也非常多，如何让一个开发人员看到一个工程名，或者jar的名称，就大概知道是干什么的，需要一个规范化的约定。  
  
例如出现pay就是支付，出现order就是下单，出现account就是用户。  
  
再如出现compose就是组合层，controller就是接口层，basic就是基础服务层。  
  
出现api就是接口定义，impl就是实现。  
  
pay-compose-api就是支付组合层接口定义。  
  
account-basic-impl就是用户基础服务层的实现。