**Soa(面向服务架构) （作业帮平台业务线）**

它将应用程序的不同功能单元（称为服务）通过这些服务之间定义良好的**接口**和契约联系起来。**接口是采用中立的方式进行定义的，它应该独立于实现服务的硬件平台、操作系统和编程语言。**

面向服务架构，它可以根据需求**通过网络对松散耦合的粗粒度**应用组件进行分布式部署、组合和使用。

SOA是一种思想 SOA的精髓是严格的松散耦合 不允许shared memory，不允许back door，不允许直接访问其它服务的数据这都好理解，因为它破坏了封装性，造成了一种内部依赖。

正因为SOA架构实现不依赖于技术，因此能够被各种不同的技术实现。

例如： SOAP, RPC REST DCOM CORBA OPC-UA Web services DDS Java RMI WCF (Microsoft's implementation of web services now forms a part of WCF) Apache Thrift SORCER 因此**REST、SOAP、RPC、RMI、DCOM**等都是SOA的一种实现而已。

Soa大部分基于webservice实现

WebService有两种方式：SOAP和REST

REST（英文：Representational State Transfer，简称REST）

Web 应用程序最重要的 REST 原则是，客户端和服务器之间的交互在请求之间是**无状态**的。从客户端到服务器的每个请求都必须包含理解请求所必需的信息。

**REST 指的是一组架构约束条件和原则。满足这些约束条件和原则的应用程序或设计就是 RESTful。**

在服务器端，应用程序状态和功能可以分为各种资源。资源是一个有趣的概念实体，它向客户端公开。资源的例子有：应用程序对象、数据库记录、算法等等。每个资源都使用 URI (Universal Resource Identifier) 得到一个唯一的地址。所有资源都共享统一的接口，以便在客户端和服务器之间传输状态。使用的是标准的 HTTP 方法，比如 GET、PUT、POST 和 DELETE。

（1）每一个**URI**代表一种资源；

（2）客户端通过四个HTTP动词，对服务器端资源进行操作，实现"表现层状态转化"。

另一个重要的 REST 原则是分层系统，这表示组件无法了解它与之交互的中间层以外的组件。

GET /zoos：列出所有动物园

POST /zoos：新建一个动物园

GET /zoos/ID：获取某个指定动物园的信息

PUT /zoos/ID：更新某个指定动物园的信息（提供该动物园的全部信息）

PATCH /zoos/ID：更新某个指定动物园的信息（提供该动物园的部分信息）

DELETE /zoos/ID：删除某个动物园

GET /zoos/ID/animals：列出某个指定动物园的所有动物

DELETE /zoos/ID/animals/ID：删除某个指定动物园的指定动物

**Soap**

RESTful Web 服务使用标准的 HTTP 方法 (GET/PUT/POST/DELETE) 来抽象所有 Web 系统的服务能力，而不同的是，SOAP 应用都通过定义自己个性化的接口方法来抽象 Web 服务.

所有的 SOAP 消息发送都使用 **HTTP POST** 方法，并且所有 SOAP 消息的 **URI** 都是一样的，这是基于 SOAP 的 Web 服务的基本实践特征。

基于xml传输

清单 5. getUserList SOAP 消息

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <?xml version="1.0" encoding="UTF-8" standalone="no"?>  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">      <soap:Body>          <p:getUserList xmlns:p="http://www.exmaple.com"/>      </soap:Body>  </soap:Envelope> |

客户端将使用 HTTP 的 POST 方法，将上述的 SOAP 消息发送至 http://localhost:8182/v1/soap/servlet/messagerouterURI，SOAP SERVER 收到该 HTTP POST 请求，通过解码 SOAP 消息确定需要调用 getUserList 方法完成该 WEB 服务调用，返回如下的响应：

清单 6. getUserListResponse 消息

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <?xml version="1.0" encoding="UTF-8" standalone="no"?>  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">      <soap:Body>              <p:get                  UserListResponse xmlns:p="http://www.exmaple.com">                  <Users>                  <username>tester<username>                  <username>tester1<username>                  ......                  </Users>                  <p: getUserListResponse >      </soap:Body>  </soap:Envelope> |

CAP原理

分布式系统的CAP理论：理论首先把分布式系统中的三个特性进行了如下归纳：

一致性（C）：在分布式系统中的所有数据备份，在同一时刻是否同样的值。（等同于所有节点访问同一份最新的数据副本）

可用性（A）：在集群中一部分节点故障后，集群整体是否还能响应客户端的读写请求。（对数据更新具备高可用性）

分区容错性（P）：以实际效果而言，分区相当于对通信的时限要求。系统如果不能在时限内达成数据一致性，就意味着发生了分区的情况，必须就当前操作在C和A之间做出选择。

(子节点网络不连通，系统如果不能在时限内达成数据一致性叫做分区，当分区出现就要C和A里选择)

CAP理论就是说在分布式存储系统中，最多只能实现上面的两点。

BASE就是为了解决关系数据库强一致性引起的问题而引起的可用性降低而提出的解决方案。 BASE是下面三个术语的缩写： 基本可用（Basically Available） 软状态（Soft state） 最终一致（Eventually consistent）

Paxos算法：<https://www.zhihu.com/question/19787937>

微服务架构

微服务架构可以说是soa的一种，是去ESB的SOA。

两者区别主要为服务的管理集中还是独立。

（但是有ESB总线（ESB提供了网络中最基本的连接中枢）的SOA架构只出现在传统企业。一般互联网企业的SOA架构都是分布式服务，所以实际的架构不会有明显分界）

分布式架构：

2pc（2PC可以在异步网络+节点宕机恢复的模型下实现一致性）

先由一方进行提议(propose)并收集其他节点的反馈(vote)，再根据反馈决定提交(commit)或中止(abort)事务。我们将提议的节点称为协调者(coordinator)，其他参与决议节点称为参与者(participants)

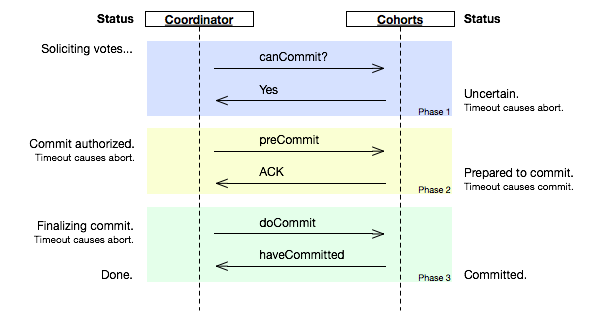
1. 协调者向所有的参与者发送事务内容，询问是否可以执行事务提交操作，并开始等待各参与者的响应.
2. 各参与者节点执行事务操作，并将Undo和Redo信息计入事务日志中.如果参与者成功执行了事务操作，那么就反馈给协调者Yes响应，表示事务可以执行；如果参与者没有成功执行事务，那么就反馈给协调者No响应，表示事务不可以执行。.
3. 送提交请求协调者向所有参与者节点发出Commit请求
4. 任何一个参与者反馈了No响应，或者在等待超时之后，协调者尚无法接收到所有参与者的反馈响应，那么就会中断事务。协调者向所有参与者节点发出Rollback请求

问题：

发生了局部网络异常或协调者在尚未发完commit请求之前自身发生了崩溃，导致最终只有部分参与者接收到了commit请求，于是这部分参与者执行事务提交，而没收到commit请求的参与者则无法进行事务提交，于是整个分布式系统出现了数据不一致性现象。

同步阻塞会极大地限制分布式系统的性能。在二阶段提交的执行过程中，所有参与该事务操作的逻辑都处于阻塞状态，各个参与者在等待其他参与者响应的过程中，将无法进行其他任何操作。

3pc:（解决2PC的阻塞，但还是可能造成数据不一致）



Paxos:

基于消息传递且具有高度容错性的一致性算法。Paxos算法要解决的问题就是如何在可能发生几起宕机或网络异常的分布式系统中，快速且正确地在集群内部对某个数据的值达成一致，并且保证不论发生以上任何异常，都不会破坏整个系统的一致性。