# 概述

Restful简化了web service的设计，它不再需要wsdl，也不再需要SOAP协议，而是通过最简单的http协议传输数据（包括xml或者json）。这样既简化了设计，也减少了网络传输量（因为只传输代表数据的xml或json，没有额外的xml包装）

# 选择

我们在设计底层服务的时候到底应该选择目前最流行的Restful架构还是选择比较老牌的webservice呢？

首先，我们需要了解的是，Restful是一种架构风格，其核心是面向资源，而erbservice底层是SOAP协议，其主要核心是面向活动。

## SOAP

其实SOAP最早是针对RPC的一种解决方案，简单对象传输协议，很轻量，同时作为应用协议可以基于多种传输协议来传递消息（http、SMTP等）。但是，随着SOAP作为webservice的广泛应用，不断增加附加的内容，使得现在开发人员觉得soap很重，使用门槛高，在soap后续发展过程中，WS-\*一系列协议的指定，增加了soap的成熟度，也给soap增加了负担。

## REST

REST并不是什么协议也不是什么标准，而是对http协议的设计初衷做了解释，在http协议被广泛利用的今天，越来越多的是将其作为传输协议，而非原先设计者考虑的应用协议。SOAP类型的webservice就是最好的例子，以至于对于http协议中的各种参数（例如编码、错误码等）都置之不顾。其实，最轻量级的应用协议就是http协议，http协议多抽象的是get、post、put、delete就好比数据库中最基本的增删改查操作，而互联网上的各种资源就好比数据库的记录，对于各种资源的操作最后总是能抽象成为这四种操作。在定义了定位资源的规则以后，开发者也会受益于这种轻量级的协议。

REST专门针对网络应用设计和开发方式，以降低开发的复杂性，提高系统的可伸缩性，rest提出设计概念和准则是：

1. 网络上所有的事物都可以被抽象为资源（resource）
2. 每一个资源都有唯一的资源标识（resource identifier），对资源的操作不会改变这些标识
3. 所有的操作都是无状态的

REST简化开发，其架构遵循CRUD原则，该原则告诉我们对于资源（包括网络资源）只需要四种行为：创建、获取、更新和删除就可以完成相关的操作和处理。我们可以通过使用统一资源标记符（Universal Resource Identifier）来识别和定位资源，并且针对这些资源而执行的操作是通过http规范定义的。其核心操作只有GET、PUT、POST、DELETE。由于rest强制所有的操作都必须是stateless的，这就没有上下文的约束，如果做分布式，集群都不需要考虑上下文和回话保持的问题，及大地提高系统的可伸缩性。

SOAP webservice有严格的规范和标准，包括安全、事务等各个方面的内容，同时soap强调操作方法和操作对象的分离，有wsdl文件规范和xsd文件分别对其定义。

如果从这个意义上讲，是否使用rest就需要考虑资源本身的抽象和识别是否困难，如果本身就是简单的类似增删改查的业务操作，那么抽象资源就比较容易，而对于复杂的业务活动抽象资源并不是一个简单的事情。比如校验用户等级、转账、事务处理等，这些往往并不容易简单的抽象为资源。

其次，如果有严格的规范或者标准定义要求，而且前期规范标准需要指导多个业务系统集成和开发的时候，soap风格由于有清晰的规范标准定义是明显有优势的。我们可以在开始和实现之前就严格定义相关接口的方法和接口传输的数据。（很多情况下是为了兼容以前项目且前台调用逻辑代码都不能动的前提下，更改底层调用的，一般就需要使用webservice模式开发，因为老的代码中已经有了明确的方法定义以及参数模型、个数等声明）

简单数据操作，无事务处理，开发和调试简单这些事使用rest风格的优势，而对于较为复杂的面向活动的服务，如果还是shiyongrest，很多时候仍然是传统的面向活动的思想通过转换工具再转换为rest服务，这种方式是没有意义的。

# 实践