共勉：

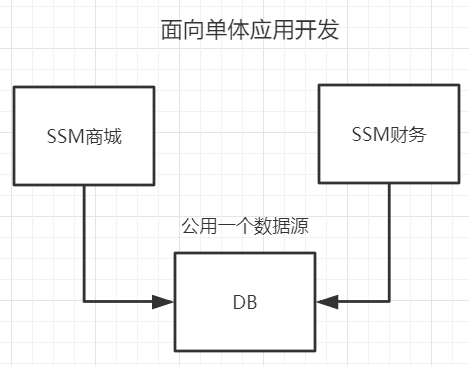


# 微服务

微服务的产生是基于服务的，有一个词叫SOA。

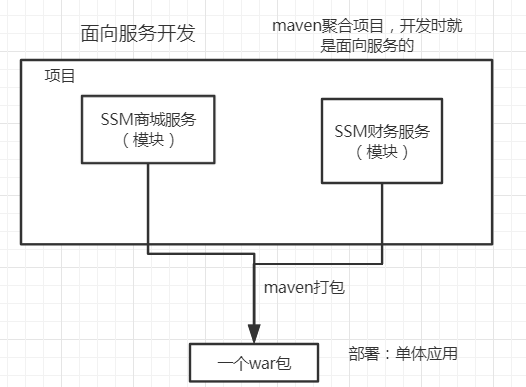
## 最早单体应用

原来的SSM项目，比如一个SSM商城是一个项目，SSM财务又是一个项目。他俩是没办法直接关联的，假设商城产生一些订单等跟钱相关的东西，财务系统要记账，在最初阶段，他来公用一个数据库，都往里面写，这样两个系统就达到交互的目的。但这样问题非常多：商城写数据的时候，财务系统就动不了；财务把某个数据删了，商城那边就没了，卧槽。



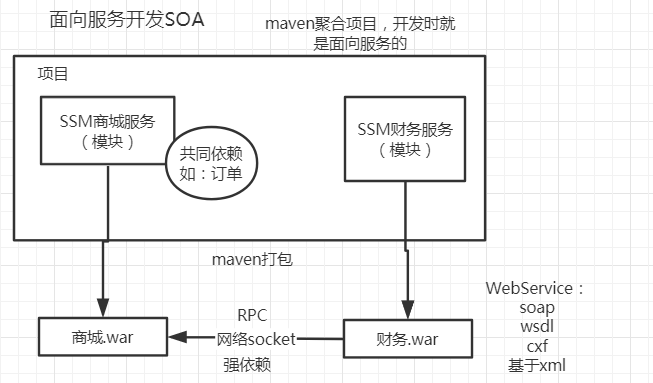
## 面向服务开发

商城、财务不再是一个独立的项目，用mave聚合项目，他们都变成了一个模块，共同组成了一个完整的项目。开发的时候，是面向服务开发的。打包的时候，用maven将其打包为一个war包，本质上还是一个单体应用。



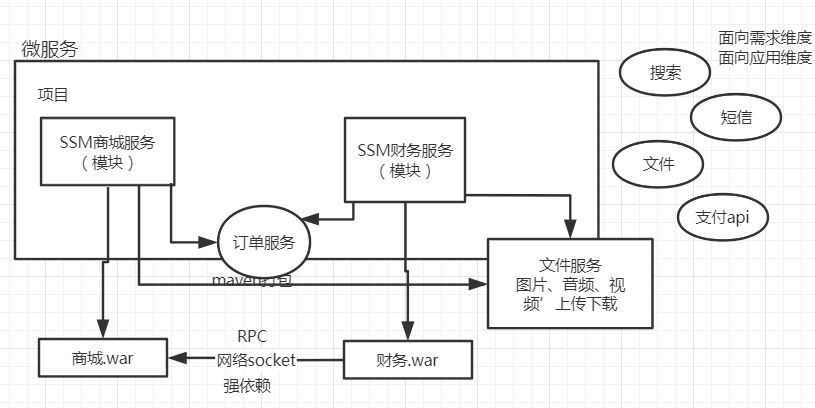
## 真正的SOA

在上面maven聚合项目上升级，商城打成一个独立的商城.war，财务打成一个独立的 财务.war ，这两个war包就跑在了两个tomcat上。此时商城和财务如果有一些共同的依赖，比如订单，两者都需要调用，但是订单打到了商城.war包里，项目启动，此时如果财务想调用订单service，此时已经调不了了，已经跨了网络了。这时候就产生了RPC（远程服务调用）。财务模块通过网络去商城war包里调用公用的模块，需要开Socket。最早是WebService（soap，wsdl，cxf），调用过程中就会产生各种问题（本质上就是发socket请求）：一次没调通怎办？两次没调通怎么办？商城挂了，财务系统再调，就调不了了（强依赖），而且数据是基于XML的，xml比较安全，但是解析麻烦。



## 微服务

把公共的服务都拆出来，这个粒度不好把控，全靠经验。微服务第一次拆比较爽，慢慢的可能微服务会变胖，变胖后要想重构？就是个灾难。所以微服务的划分全靠经验。而且微服务多了之后，就会不好管理。



比如支付api，可以把支付宝、微信支付都做在一起，统称支付api，也好拆。如果以后支付做的大了，或者说调用不平均了，支付宝支付的多，微信的没人用，那么做负载均衡的时候，就可以把支付宝多复制出来几份，这就要提到了微服务的负载均衡。

（有的微服务只是service，有个微服务有View）

详细看下图：

