《01\_我们为什么要使用复杂的微服务架构？》

单块应用：在intellij idea/eclipse建一个工程，spring mvc+spring+mybatis整合，里面写一堆controller、service、dao、mapper、sql，加一大堆的配置文件，有可能还会引入一些redis、elasticsearch、mq之类的一些依赖，可能会操作一些类似的东西

maven插件，把工程里的所有代码和依赖打包成一个jar包/war包，配置文件

公司提供给你一台linux服务器，虚拟机/物理机，在机器上你自己先部署了一个tomcat，然后把你写好的系统的jar包/war包放到tomcat指定目录下去，然后重启一下tomcat，tomcat一旦启动就会监听类似于8080的端口号

然后你针对8080 端口号发起http请求，请求会由tomcat直接转交给你的spring mvc，一层一层调用你写的代码

单块应用的问题，10个人以上维护一个系统，频繁的代码分支进行合并，冲突很多，因为人多，导致很混乱，谁也不能保证10个人以上还能每个人就一定只会更新自己负责的一部分代码

每次光是解决大量的代码冲突，就会耗费好几天的时间，解决完了冲突，还得进行一下测试，保证代码正常运行

完全可能会你拉一个分支修改了代码之后，别人已经在你上线之前就更新了别的分支的代码，合并到master分支还发布上线了，所以你上线之前一般需要把master分支最新代码合并到一个测试分支上，把你的功能分支再合并上去，然后进行测试

每次测试，可能都会因为别人频繁改动代码，导致必须把系统所有相关功能都测试一遍，而且很可能因为别人胡乱改过你负责的部分的代码，导致合并之后，测试的时候运行就有问题，这是完全可能的

所以这是一种严重耦合的情况，会导致每次测试和上线，都需要大量的成本解决代码冲突，因为别人可能胡乱修改你的代码，导致测试需要全量回归所有功能，导致上线很慢很麻烦，而且更麻烦的是不同版本之间的上线协同

可能你拉的一个代码分支，修改的仅仅是针对5个功能，但是另外一个人之前修改了20个功能，你们的代码一旦合并了，可能导致大量的地方都有变动，可能会导致另外几十个依赖你们的代码的功能也会有一定的潜在的风险

在这种单块系统的情况下，代码一旦合并，上线之前，必须在测试环境对上百个完整的功能都做一下回归测试，这个耗费的时间就很长了

可能有的人要在某一天上线，你就必须等待他后一天再上线，每个人都必须做上线前做代码合并，然后全量回归，最后再上线，下一个人继续合并最新代码，全量回归，再上线，这个过程很麻烦，一旦出了差错，可能导致你直接把代码合并到master部署，上线可能会失败的

1. 很多人维护一个单块应用，频繁的进行代码合并，频繁的解决代码冲突，解决冲突的时间和成本很高的，导致开发效率低下
2. 每次上线都要跟最新代码进行合并，重新进行全量功能的回归测试，很多代码都可能给有变动，必须全量回归测试，耗费时间很多，开发效率低下
3. 多人频繁上线，你等我，我等你，互相协调困难，而且可能会出现别人多次先上线，你多次重复的合并代码，解决冲突，全量回归测试，做很多次重复的事情，导致效率低下
4. 测试服务器都很少，就一台测试服务器，你开发好了代码，你连测试都不能测，必须等待别人的分支先测试完毕上线，你才能去合并代码，解决冲突，等到一个测试环境可以用，回归测试
5. 10个人以内维护一个单块应用，基本这些成本还不算太大，但是一旦10人以上维护一个单块应用，上述的成本会极大，导致系统每个需求的测试和上线，都非常缓慢，要耗费大量时间做全量回归测试，上线日期还得互相配合互相等待互相协调，一个疏忽，就可能导致没侧测试完全的代码上线出线上事故
6. 如果你想要升级技术架构，不能随意升级，因为可能影响别人，你得让所有人都学习这个新技术架构才行；如果你想要升级已有技术的版本，不能随意升级，因为可能新版本对你的代码没影响，但是对别人的代码有影响

10人以内基本都问题不大，但是10人以上，往往维护一个单块应用就比较麻烦了，通常建议的比较合理的是5人以内的小团队负责维护一个系统，这对这个问题，就需要进行微服务架构，把大系统拆分为很多小系统，几个人负责一个服务

这样每个服务独立的开发、测试和上线，代码冲突少了，每次上线就回归测试自己的一个服务即可，测试速度快了，上线是独立的，只要向后兼容接口就行了，不需要跟别人等待和协调，技术架构和技术版本的升级，几个人ok就行，成本降低，更加灵活了