0827-早上看了一篇单体应用转分布式的一个小节点

先总结下昨天晚上看的多线程，里面也有提到分布式、微服务

分布式：

就是把我们的单体应用copy一份放到单独的服务器上运行，以应对高并发时的场景

微服务：

就是当我们的系统越来越大时，服务也就越来越多，分布式里每个系统的模块都有自己的职责，而业务层肯定是占资源最多的，而相应的每个业务模块请求次数是不一样的，也就是重要性不一样，所以可以把重要的模块，占用资源多的模块单独拆分出来，部署到服务器上，以更好的应对高并发场景。

会遇到的问题：

如何分发请求呢？（这就用到了负载均衡服务器）

中间就需要一个负载均衡的一个系统来分发我们的请求，让用户的请求能够尽可能的均匀分发到分布式中的系统中、均匀分发到微服务

Nginx、Apche（在..第七层,应用于应用层，比如Tomcat）

Lvs、hbxxx（在第四层，应用较广，主要用于如：数据库）

F5（是个硬件，价格较贵，很少公司用）

我们一般使用的就是Nginx 这个负载均衡、反向代理服务器（很好用，简单，配置就行）

Nginx 默认是采用轮询的方式将请求分发到服务器

会遇到的问题：

单点故障（在执行系统时，登录不进去）（因为session 存储机制问题）

解决session 一致性方法：

session粘贴（比如Nginx的Ip\_hash算法）

session复制（采用组播的通讯机制，是给系统配置集群版tomca，实现简单，但是每个session

都得在每个Tomcat上copy一份，占用资源）

session共享（运用Redis（缓存中间件，让Redis管理session，实现非常简单，引入

spring-session-Redis 依赖，给每个系统配置Redis，启动类加注解EnableRedisHttpSession就可以了）；

用的最多的，也就是第二和第三种，但三种为更简单，实现也很容易，

除了session共享分布式和微服务里还是会遇到各种各样的问题：

比如说昨天晚上看的千万级、亿级的并发量，怎么去解决：

（方法：**请求合并**）：

思路：

所谓的高并发，其实还是查询的占大多数，所以我们可以把一段时间内大量相似的请求合并成一个请求去库里面查询出来，再返回到请求池中，也就是各个用户

问题：

怎么控制一段时间内，怎么合并到一起，就算合并到一起了怎么在返回到各个用户手里呢？

解决：

1. 用类的初始化方法：init（）这个类只要被初始化就将用定时任务的线程池，开出来一个单独的定时线程，去每隔10m后者更多的时间去执行合并请求
2. 将这段时间内的请求的参数存储起来，装到一个集合里
3. 利用每个请求为一个线程，把请求（线程）和上面的线程联系起来，用的就是多线程间的通信机制，用callable、futrue实现的，再利用其请求的id也就是参数不一样，根据参数和线程返回给各个用户