有单个服务实例并发处理上限限制的负载分发任务。

自研TTS服务用于替代厂商的TTS， 但自研TTS服务存在单实例并发上限。

记录服务实例以及其并发上限。 -》服务注册-> spring cloud eureka.

假设每个服务实例的并发上限是一致的。 ->对服务类别配置上报并发上限数量。

1. Apache ignite框架：

提供了分布式数据结构， 其中有分布式semaphore。

该场景天然适用semaphore， 所以首先考虑的是这个方案。

* 1. 整体服务使用一个总的Semaphore， 每个服务实例再有各自一个小的semaphore。 首先获取总的semaphore，若获取成功再获取小的semaphore。 坏处：小的semaphore集合由什么数据结构维护： 1. List， 一个获取失败后如何处理？-》延迟严重 2. 排序结构？ 该数据结构不支持 。 第二个坏处还需要保障总的semaphore与小的semaphore之间的一致性； 并且semaphore不能动态增加（新服务实例注册则无法处理）。
  2. 代理模式：每次接受请求和处理完成后分别减少、增加该实例的semaphore。由于TTS服务是python写的， 本人对python不熟且时间紧迫，因此放弃。
  3. Python端做并发控制处理： 按理说应该不难，使用randomrule+服务实例达到上限时reject策略即可， 但tts服务同事也对这个不熟+领导要求做负载分发服务。

1.4：参考令牌算法：

Spring cloud有一个loadbalance concurrency limit的包，方法已经废弃，参考其中源码，使用的是令牌算法： 即每秒生成固定数量的令牌， 每个请求来了之后先获取令牌， 拿不到令牌的请求应该被拒绝以达到限制每秒qps的作用。

服务实例注册时上报的并发数量作为该实例的可用令牌，请求来了时候获取令牌， 处理完成之后放回。

上报服务实例id: ip:port:serviceName:pid

需要一个中间组件： 因为eureka是AP模式， 所以不能依靠eureka。使用redis来做令牌管理。 且redis list支持block处理。 性能为一次网络IO， list取右，入左(O(1))。 服务实例下线时(遍历令牌列表剔除令牌-O(n))。使用redis lua脚本来保障操作事务性。

\*\* redis配置通常也不是强一制性， 持久化通常使用aof，且策略使用every seconds，所以可能丢失一秒的数据。