说一下自己公司的服务注册中心怎么技术选型的？生产环境中应该怎么优化？

1. 画出公司的服务注册中心原理图！

2. 考虑一下数据一致性

3. 看一下公司的服务注册、故障发现的时效性是多长？

4. 注册中心最大能支撑多少服务实例？

5. 生产环境是如何部署的？几台机器？每台机器的配置？

6. 如何保证的可用性？服务注册、故障、发现的时效性是否有做过优化？

**优秀作业1:**

学员:顾飞，

1、选型：我们公司使用的Spring Cloud服务框架，所以注册中心采用Eureka，Eureka选择了CAP中的AP，也就保证牺牲强一致性保证可用性。

由于是peer-to-peer方式，所有Eureka地位都是对等的，平等对外提供服务，服务往一个Eureka节点注册时，会异步同步给集群中其他Eureka节点。

如果同步失败，也不会影响可用性，会出现短暂的数据不一致。直到同步心跳信息时，发现数据不一致，然后重新同步缺失的注册信息，实现最终一致性。

2、问题优化：开始采用默认的参数的时候，遇到一些问题，发现服务上线延时不稳定，有时会1分钟以上，而服务下线延时更长，最高会达到几分钟，时效性太差！

后来我优化了相关参数，现在将上线延时控制和下线延迟控制在秒级。时效性提高了10倍，虽然Eureka请求压力大了10倍，但是依然很轻松。

①、上线部分：因为时效性主要是：ReadOnly缓存的默认定时同步间隔是30秒 + 服务拉取注册表默认30秒 + Ribbon更新本地缓存一个时间，所以极端下，延时超过1分钟。所以我把上图中2处，ReadOnly缓存的默认定时同步间隔改成3秒；服务拉取时间改为3秒。这样实现秒级上线，几秒钟。

②、下线部分：因为下线实现性主要是 线程定时检测服务心跳故障的时间间隔，默认60秒（另外认定心跳过期的时间，默认90秒）+ ReadOnly缓存的默认定时同步间隔是30秒+服务拉取注册表默认30秒，极端情况下，延时了几分钟了。

所以我把上图1处线程定时检测服务心跳故障的时间改为6秒；认定心跳过期时间改为9秒；心跳间隔改为3秒。这样实现服务下线秒级感知， 极端情况下最高也只有十几秒。然后关闭自我保护机制，避免了其代码中的bug。

3、规模：因为系统每天QPS只有300-400，部署了2个节点，4核8G配置。并发、容量、可用性都可以保证。

4、容量计算：假如系统是几百个服务，2000个实例的大型规模系统，默认拉取注册表间隔为3秒，心跳时间为3秒，这样Eureka压力为每秒1333次请求

由于Eureka是纯内存操作，并且多级缓存避免并发读写冲突，并且多节点分担读负载，所以使用高性能的服务器还是可以支撑的。

但是由于Eureka集群是点对点模式，服务和心跳信息要进行集群同步，多节点可以分担服务发现负载，但是不能分担服务注册和心跳负载。

综合考虑，最好还是控制在1000节点内，配合上述参数优化，可以实现秒级上下线感知。

老师您看，我这个项目经历能符合互联网高级工程师吗？P5的。硕士，2年1个月工作经历。

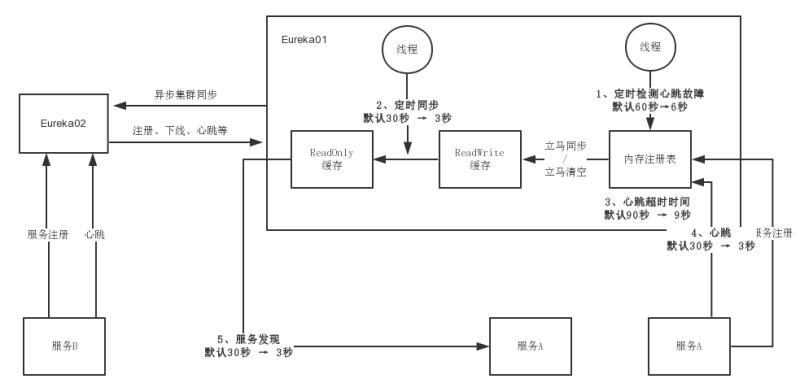
【A2P短信网关，项目上线4个月，日均短信量600万，Elasticsearch积累了6亿数据量220G，ES使用5台6核64G机器，部署5个shard，2个副本。ES堆内存分配20G，剩下40G留给文件系统缓存，将数据全部缓存到os cache，实现毫秒级搜索。】

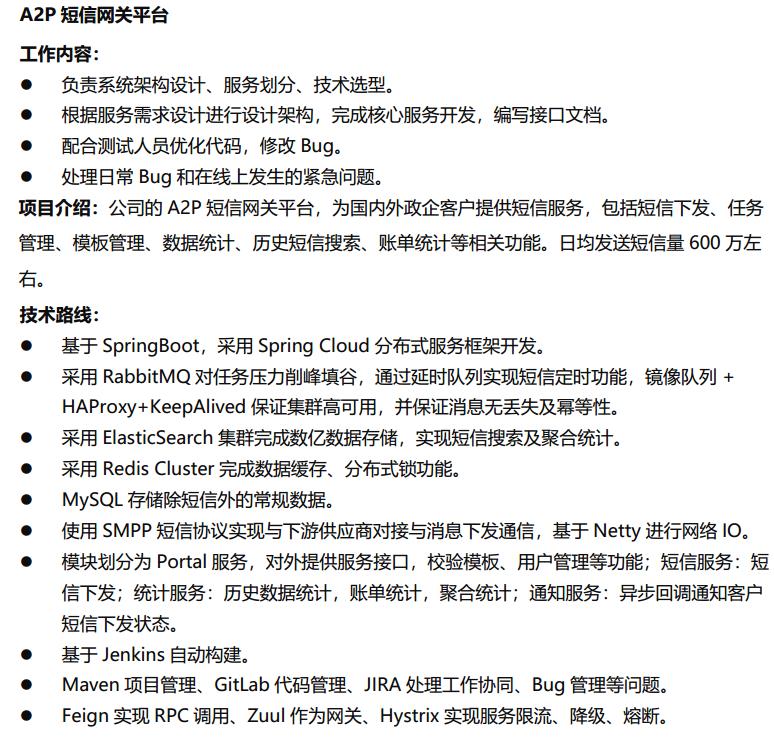
【RabbitMQ3台镜像集群，4核16G】 【集群部署了2个Eureka、2个Zuul、3个Portal服务、6个短信发送服务、3个通知服务，都是4核16G内存。】

老师别笑话我啊，传统通信公司，是在没有太大的QPS，写项目经历都愁。。。这个量已经是在吹牛了，不知道吹的靠不靠谱？有啥漏洞吗？

点评：

很不错。如果可以把训练营的一些技术融入到项目中，没问题的！





**优秀作业2:**

学员：carson

1、画出公司的服务注册中心原理图！

公司采用zookeeper 集群部署，机器都是双核四G50G硬盘，可能配置一般，但是独立运行一个zk没问题吧

1：Zookeeper leader和follower两种角色，公司在三台机器部署了三个zk，一个leader，2个follower

2：leader负责注册服务，然后把数据同步给follower

3：读取服务的时候可以通过zookeeper和foliower读取，follower和leader保证数据强一致性

4：调用放监听zookeeper服务，一旦服务变更，zk会同步给服务更新注册列表信息到本地缓存

数据一致性方面：

1：zookeeper是CP模型，leader节点接收来自服务的所有数据，然后同步给follower节点（ms级别）；

2：follower节点不接收，数据来自leader节点的同步。

3：当leader节点故障的时候，会重新选举leader节点，这个时候zookeeper服务不可用，服务不能注册或者发现下线的服务，可能会停顿ms。

4：当follower选取成leader时候，那么这个时候新的服务又可以注册，然后leader会将服务同步给follower,数据一致性很快得到保障,消费方面很快能够发现新上线的服务。

高可用性方面：

zookeeper集群毕竟是CP模型,可用性理论上很难保障,一般zk故障基本会引起可用性，导致线上出现故障

比如磁盘满了，zk故障，一般运维会做清磁盘处理或者扩容处理或者重启操作。

时效性方面：

zookeeper 目前没啥问题,正常情况很快可以感知服务的上线，服务的下线。

容量方面：

这块可以和高可用结合起来，确实容量出现问题，zk容易出现故障。目前的话大概有几百个实例向zk注册服务。

再来梳理下另外一个注册中心,Eureka 虽然咱们没有使用 Eureka 集群模式

1：有多个注册中心 Eureka集群部署Eureka-A、Eureka-B

2：有个服务需要启动向Eureka-A注册服务，Eureka-A会将注册表信息同步给Eureka-B

3：Eureka-A、Eureka-B 是peer-To-peer模式，同步的注册信息都是一致的

4：服务会不断的发送给注册中心心跳，集群里任何一个Euerka实例接收到写请求之后，会自动同步给其他所有的Eureka实例。

数据一致性方面：

1：Eureka是peer模式，每个服务实例都是对等的，实例直接会不断同步数据。

2：当其中Eureka实例故障的时候，可以从其他Eureka拉去服务注册表，但是这个过程中服务注册信息可能不是最新的，不能保证强一致性，但是保证最终一致性。

时效性方面 eureka，默认配置非常坑，服务注册心跳需要30s，定时扫描注册中心需要60s，如果扫描的服务发现90s内没有心跳，发现实例需要移除， 定时扫描writeCache同步给readOnly需要30s，消费方定时拉去注册信息需要30s。

所以真实环境为了保证时效需要对参数进行调优。 比如这些参数： eureka.server.responseCacheUpdateIntervalMs = 3000 eureka.client.registryFetchIntervalSeconds = 3000 eureka.client.leaseRenewalIntervalInSeconds = 3 eureka.server.evictionIntervalTimerInMs=3000 eureka.instance.leaseExpirationDurationInSeconds = 6

容量方面：

也很难支撑大规模的服务实例，因为每个eureka实例都要接受所有的请求，实例多了压力太大，扛不住，也很难到几千服务实例。

提问： 老师，您好，有个问题关于zk可用性的。 zookeeper高可用很难保障，线上目前集群部署leader/follower方式，比如leader节点故障了，可能是由于磁盘满了导致故障， 那么如何保障不重启，清磁盘的方式来重新使用zk服务，这样可用性很难保障啊？

问题解答：

zk可用性本来就是很难保证的，面试训练营里分析了，他核心保证的是一致性而不是可用性，所以很多大公司都经常因为zk故障导致依赖他的各种系统都崩溃

