Eureka

在之前我们学习了基于dubbo的zookeper注册中心

学习思考

1. zookeeper与eureka的不同
2. eureka的心跳机制
3. 基于eureka的远程调用是怎么创建的 , 原理是什么
4. 为什么需要注册中心
5. 主流的注册中心
6. 注册中心 消费者的自我保护机制
7. 注册中心 , 服务与监控的流程

注册中心

这次eureka和zookeeper注册中心也差不太多 , 都是类似于一个平台, 管理消费者与生产者之间的服务 , 作用都是提供服务注册与发现功能，对服务的url地址进行统一管理.

1.zookeeper与eureka的不同

eureka与zookeeper在本质上的功能是没有什么不同的, 不过eureka是基于springCloud的框架 , 而zookeper是基于dubbo的框架 , 所以从应用的角度来讲 , 使用eureka会更加方便一点

1.1 两个框架的远程访问服务获取数据的方式是不一样的

zookeeper的底层是通过生成一个动态代理类 , 然后在动态代理中 , 通过 rpc协议来进行数据的访问而eureka的底层也是通过生成动态代理 , 不过不同的是eureka的底层是通过**http**的访问方式来进行数据的访问 (相当于暴露一个接口 , 当请求访问的时候只需要提供接口内的返回数据即可) , 服务于服务之间的调用 , 两者在很大程度上有着本质的区别

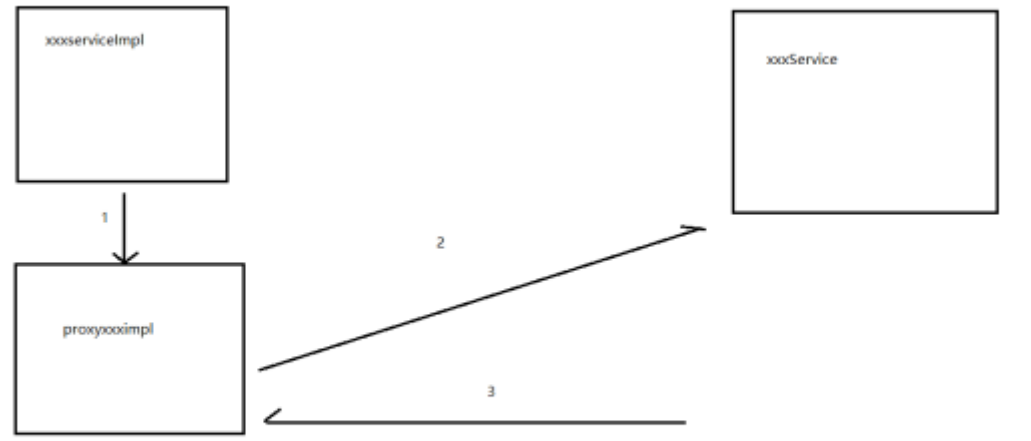
1.2 性能

在性能方面很明显zookeper的底层性能是更高的 , 因为rpc协议是更加底层的东西 , 是控制中心的内部通过拼接地址来进行远程访问的 , 不过**缺点**也很明显 , 无法进行外部的访问 , 或者说在不同语言开发的微服务项目下, zookeper就完全失去了优势 , 而eureka虽然失去了一点性能上的问题 , 不过随之带来的是它的**高可用** , **易开发**的优点 , 而且相比于zookeper的内部访问方式 , 通过外部暴露接口 , 不进可以远程调用 ,内部调用也是可以的

图解 :

dubbo

dubbo

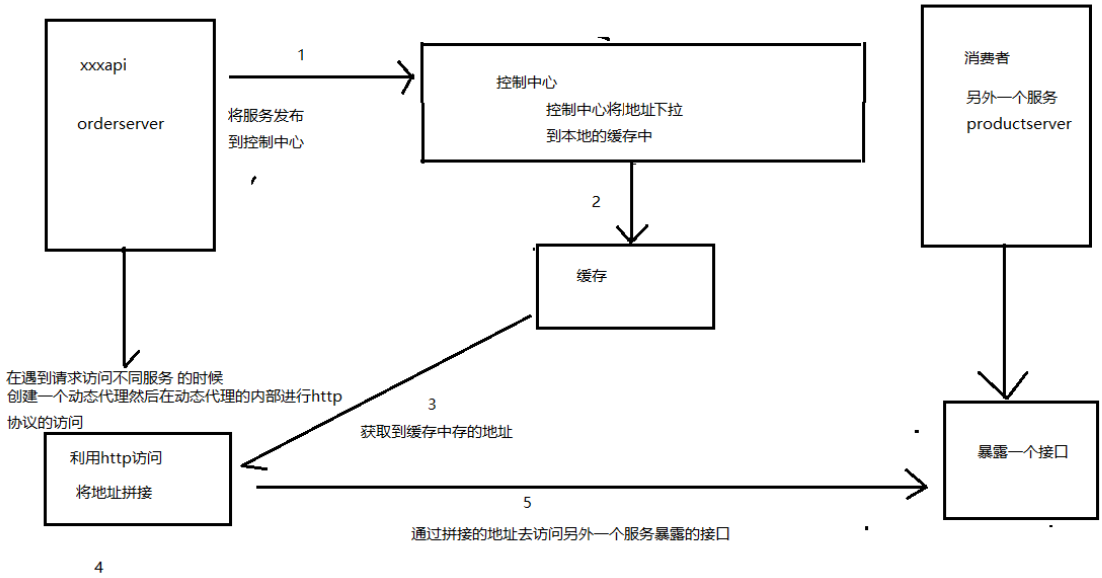


在访问服务过后 服务端将数据以二进制的方式返回则需要序列化

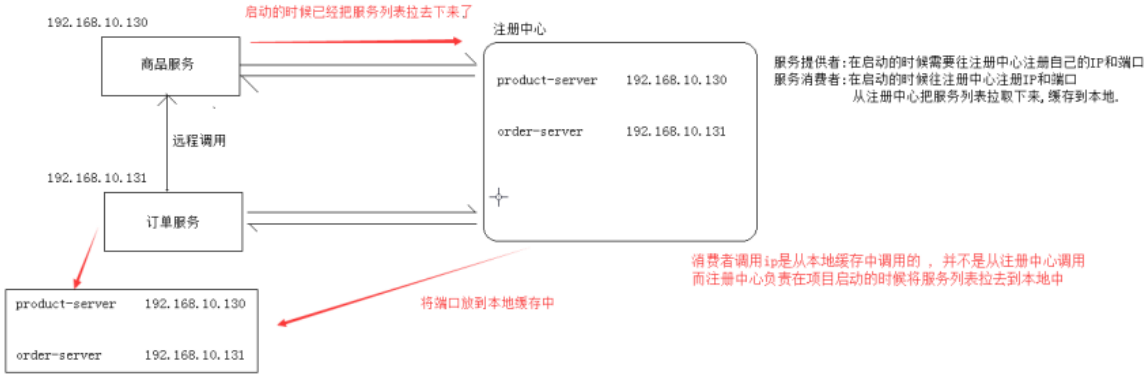
底层是先通过动态代理的方式先 调用动态代理的服务类
动态代理通过 sokect 与rpc协议访问到另外一个服务去

1

eureka



注册中心远程调用访问执行流程



2.eureka的心跳机制

在一般情况下，注册中心，会对注册的服务 采取 **心跳检测** 也就是说每隔30秒会检测一下服务存活的状态，如果超过了3次连接，**注册中心就自动干掉这个服务的实例**，动态添加与删除服务，我们的消费者都可以感知到，但是在网络波动的情况下，微服务是可以访问的，eureka访问不到的情况下，会自动触发保护机制

3.eureka的远程调用及其原理

当一个请求访问我们的服务时候，通过 eureka的负载均衡将我们的请求分发到注册中心上面，从注册中心的本地缓存列表获取 另外一个服务的ip地址，然后获取到ip地址后通过动态代理进行拼接，底层使用 **HttpurlConnection** 从而访问另外一个服务暴露的接口，将数据序列化回来

3.1.1配置一个注解和一个模板来实现负载均衡 与远程调用的访问

通过 RestTemplate 这个springCloud内置模板可以实现我们想要的远程调用的目的，然后可以通过配置 `@LoadBalanced` 注解来实现springCloud内置的一个负载均衡策略

详情请看代码实现篇

4.为什么要用注册中心

其实微服务之间没有注册中心也是可以运行的，不过这样带来的后果就是我们并不能及时的获取生产者生产到的最新的服务，而且没有注册中心缓存起来的 服务列表，我们远程访问将必须填写ip 写成硬编码，这样对我们的开发造成了极大的不便

5.主流的注册中心

- zookeeper √
- Eureka √
- consul
- etcd

6.注册中心的自我保护机制

6.1 自我保护机制的产生与解决办法

通过心跳机制我们可知 ，注册中心服务 在运行期间会统计心跳比例失败在15分钟内是否低于85%，如果低于85% 注册中心服务，会将这些实例保护起来，让这些不会过期，但是如果在保护期间，我们的服务提供者非正常下线了，

此时消费者就会拿到一个无效的服务实例，此时就会调用失败，对于这个问题，需要消费者端要有一个容错机制，

比如**重试，断路** 等

我们在单机测试的时候很容易满足心跳失败比例在 15 分钟之内低于 85%，这个时候就会触发 Eureka 的保护机制，一旦开启了保护机制，则服务注册中心维护的服务实例就不是那么准确了，此时我们可以使用eureka.server.enable-self-preservation=false来关闭保护机制，这样可以确保注册中心中不可用的实例被及时的剔除（不推荐）自我保护机制　自我保护机制2.0

7.注册中心服务与监控的流程

1. **Register(注册)**

   Eureka客户端将关于运行实例的信息注册到Eureka服务器。注册发生在第一次心跳, 也就是意味着我们在启动的时候就完成了服务的注册 , 与监控

2. **Renew(更新 / 续借)**

   Eureka客户端需要更新最新注册信息（续借），通过每30秒发送一次心跳。更新通知是为了告诉Eureka服务器实例仍然存活。如果服务器在90秒内没有看到更新，它会将实例从注册表中删除。建议不要更改更新间隔，因为服务器使用该信息来确定客户机与服务器之间的通信是否存在广泛传播的问题

   其实就是检测服务是否更新 , 没有更新90秒后回从实例的注册表中删除

3. **Fetch Registry（抓取注册信息)**

   Eureka客户端从服务器获取注册表信息并在本地缓存。之后，客户端使用这些信息来查找其他服务。通过在上一个获取周期和当前获取周期之间获取增量更新，这些信息会定期更新(每30秒更新一次)。获取的时候可能返回相同的实例。Eureka客户端自动处理重复信息。

4. **Cancel（取消)**

   Eureka客户端在关机时向Eureka服务器发送一个取消请求。这将从服务器的实例注册表中删除实例，从而有效地将实例从流量中取出。