Spring Cloud 相关组件

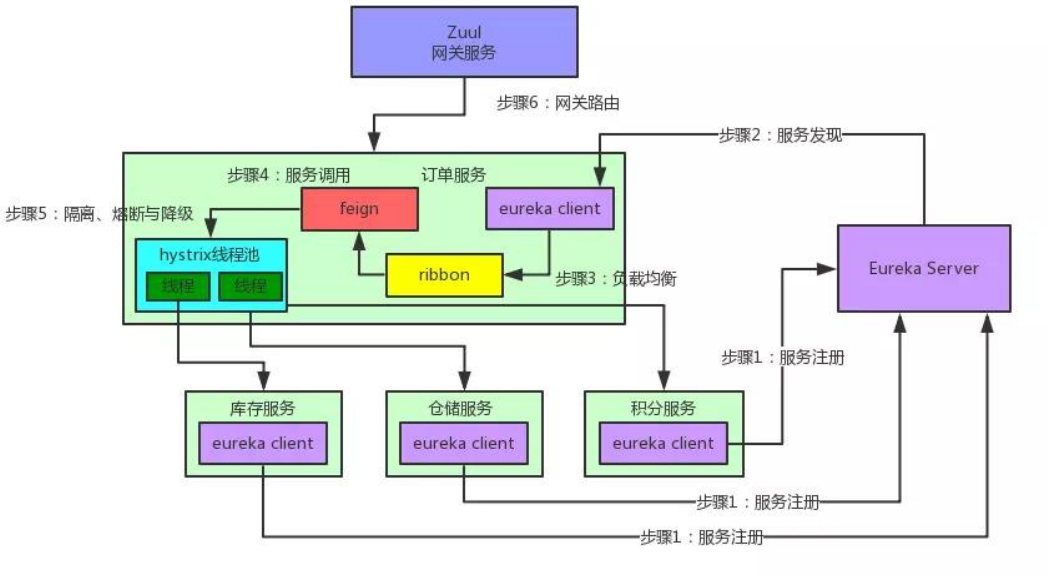
Eureka：各个服务启动时，Eureka Client 都会将服务注册到 Eureka Server，并且 Eureka Client 还可以反过来从 Eureka Server 拉取注册表，从而知道其他服务在哪里

Ribbon：服务间发起请求的时候，基于 Ribbon 做负载均衡，从一个服务的多台机器中选择一台

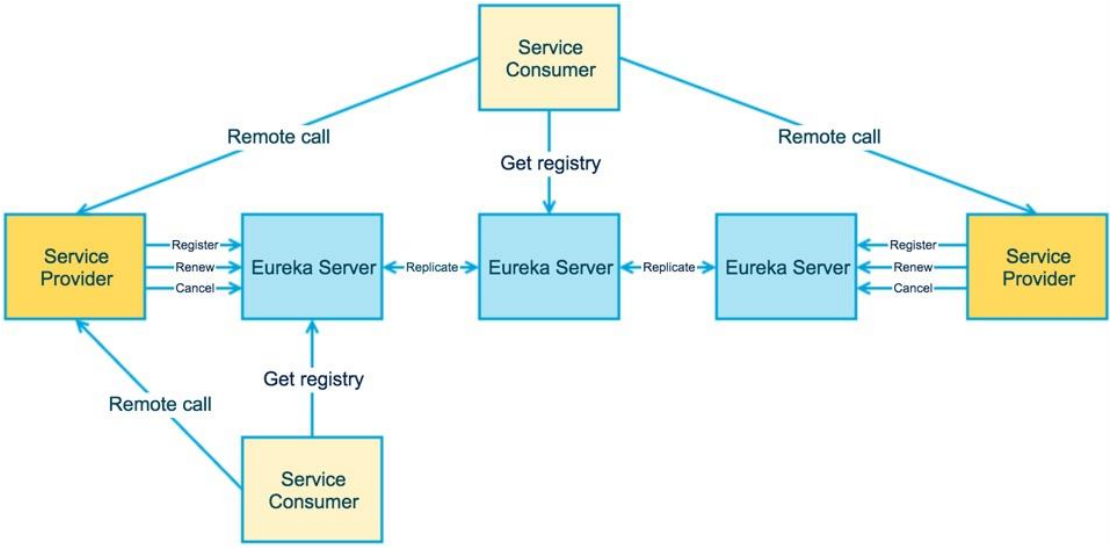
Feign：基于 Feign 的动态代理机制，根据注解和选择的机器，拼接请求 URL 地址，发起请求

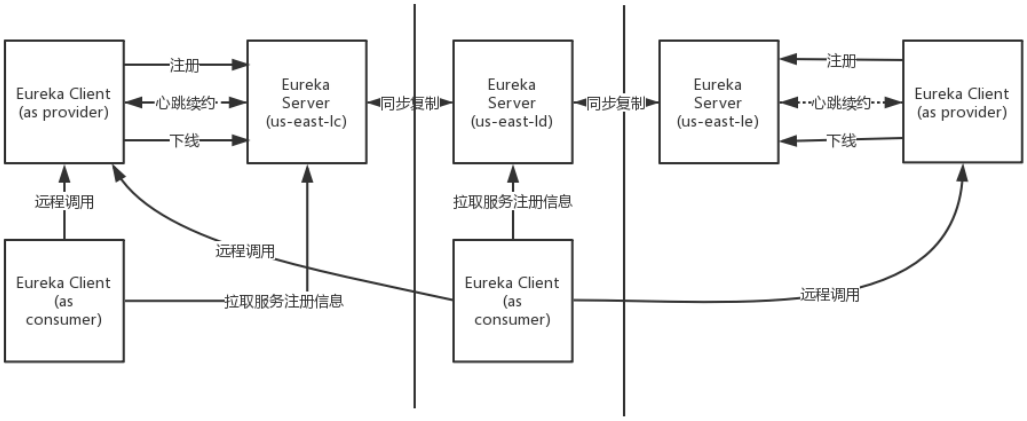
Hystrix：发起请求是通过 Hystrix 的线程池来走的，不同的服务走不同的线程池，实现了不同服务调用的隔离，避免了服务雪崩的问题

Zuul：如果前端、移动端要调用后端系统，统一从 Zuul 网关进入，由 Zuul 网关转发请求给对应的服务



服务注册与服务发现 Eureka





consumer：服务消费方，eureka client 角色，可以从 eureka server 上拉取到其他已注册服务的信息，从而根据这些信息找到自己所需的服务，然后发起远程调用。
provider：服务提供方，eureka client 角色，可以向 eureka server 上注册和更新自己的信息，当然作为 eureka client ，它也可以从 server 上获取到其他服务的信息。
Eureka server：服务注册中心，提供服务注册和服务发现功能；
同步复制 ： eureka server 之间进行注册服务信息的同步，这样可以保证集群中每个 server 都能提供完整的服务信息。

Eureka Server 进行服务治理的过程：
服务注册：”服务提供者”在启动的时候会通过发送 REST 请求的方式将自己注册到 Eureka Server 上，同时带上了自身服务的一些元数据信息（hostName 之类的）。Eureka Server 接收到这个 REST 请求之后，将元数据信息存储在一个双层结构 Map 中，其中第一层的 key 是服务名，第二层的 key 是具体服务的实例名。
服务同步：由于服务注册中心之间互相注册为服务（Eureka Server 高可用场景），当服务提供者发送注册请求到一个服务注册中心时，它会将该请求转发给集群中相连的其他注册中心， 从而实现注册中心之间的服务同步。通过服务同步，两个服务提供者的服务信息就可以通过这两台服务注册中心中的任意一台获取到。
服务续约：在注册完服务之后，“服务提供者”会维护一个心跳用来持续告诉 Eureka Sever " 我还活着 ”，以防止 Eureka Server 的 “ 剔除任务 ” 将该服务实例从服务列表中排除出去。
服务消费：当我们启动“服务消费者”的时候，它会发送一个 REST 请求给服务注册中心，来获取上面注册的服务清单 。为了性能考虑， Eureka Serer 会维护一份只读的服务清单来返回给客户端，同时该缓存清单会每隔 30 秒更新一次。
服务调用：“服务消费者”在 获取服务清单后，通过服务名可以获得具体提供服务的实例名和该实例的元数据信息。因为有这些服务实例的详细信息，所以客户端可以根据自己的需要决定具体调用哪个实例，在 Ribbon 中会默认采用轮询的方式进行调用，从而实现客户端的负载均衡。
服务下线：服务实例进行正常的关闭操作时，它会触发一个服务下线的 REST 请求给 Eureka Server，告诉服务注册中心：“我要下线了 ”。服务端在接收到请求之后，将该服务状态置为下线(DOWN)，并把该下线事件传播出去。
失效剔除：有些时候，我们的服务实例并不一定会正常下线，可能由于内存溢出、网络故障等原因使得服务不能正常工作，而服务注册中心并未收到“服务下线 ” 的请求。为了从服

务列表中将这些无法提供服务的实例剔除，Eureka Server 在启动的时候会创建一个定时任务，默认每隔 一段时间（默认为 60 秒） 将当前清单中超时（默认为 90 秒）没有续约的服务剔除出去。

自我保护：Eureka Server 在运行期间，会统计心跳失败的比例在 15 分钟之内是否低于 85%，如果出现低于的情况（在单机调试的时候很容易满足， 实际在生产环境上通常是由于网络不稳定导致），Eureka Server 会将当前的实例注册信息保护起来， 让这些实例不会过期，尽可能保护这些注册信息。

tips：Spring Cloud Eureka 实现的服务治理机制强调了 CAP 原理中的 AP，即可用性与分区容错性，它与 Zoo Keeper 这类强调 CP(一致性、分区容错性）的服务治理框架最大的区别就是，Eureka 为了实现更高的服务可用性，牺牲了一定的一致性，在极端情况下它宁愿接受故障实例也不要丢掉“健康”实例，比如，当服务注册中心的网络发生故障断开时，由于所有的服务实例无法维持续约心跳，在强调 AP 的服务治理中将会把所有服务实例都剔除掉，而 Eureka 则会因为超过 85% 的实例丢失心跳而会触发保护机制，注册中心将会保留此时的所有节点，以实现服务间依然可以进行互相调用的场景，即使其中有部分故障节点，但这样做可以继续保障大多数的服务正常消费。

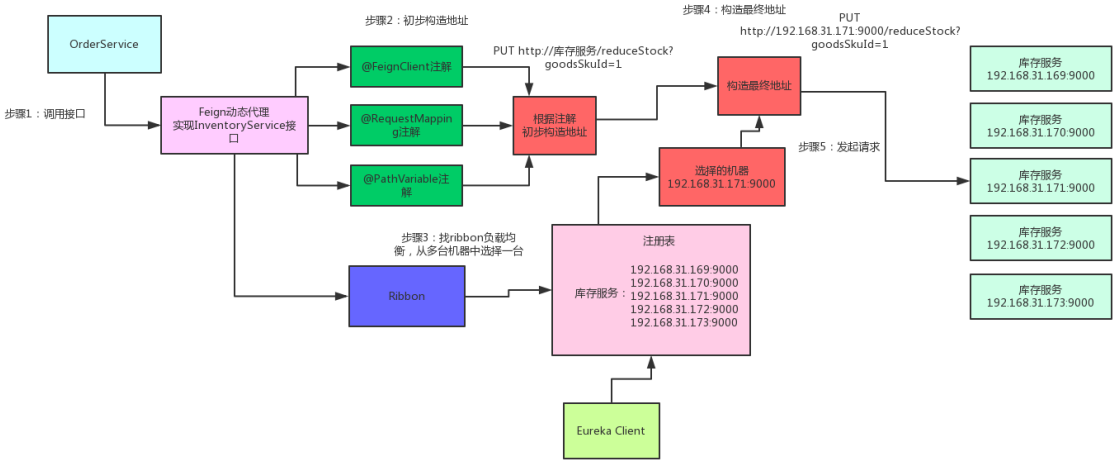
负载均衡 Ribbon

Ribbon 的作用是**负载均衡**，会帮你在每次请求时选择一台机器，均匀的把请求分发到各个机器上

Ribbon 的负载均衡默认使用的最经典的 **Round Robin 轮询算法**。

此外，Ribbon 是和 Feign 以及 Eureka 紧密协作，完成工作的，具体如下**：**

- 首先 Ribbon 会从 Eureka Client 里获取到对应的服务注册表，也就知道了所有的服务都部署在了哪些机器上，在监听哪些端口号。
- 然后 Ribbon 就可以使用默认的 Round Robin 算法，从中选择一台机器
- Feign 就会针对这台机器，构造并发起请求。

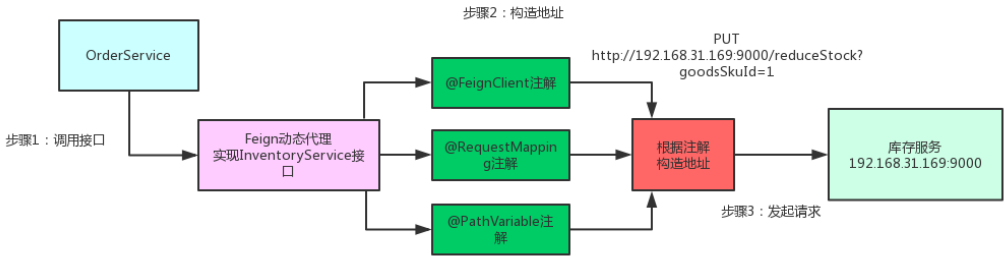


声明式 REST 调用 Feign

Feign 的一个关键机制就是使用了动态代理

- 首先，如果你对某个接口定义了@FeignClient 注解，Feign 就会针对这个接口创建一个动态代理

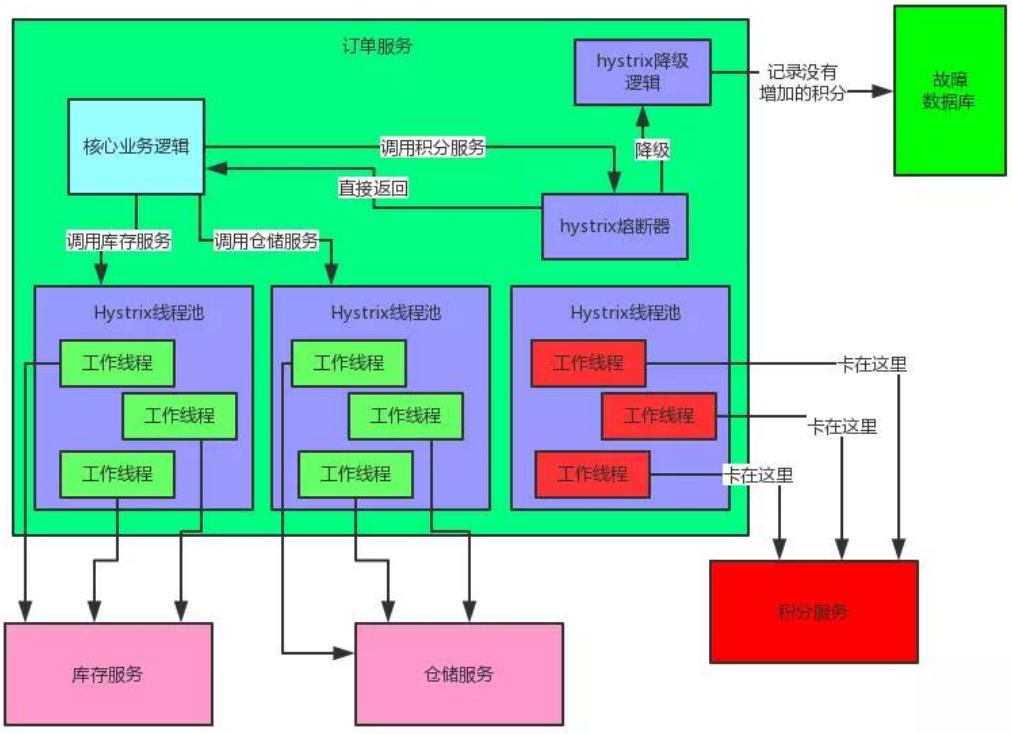
- 接着你要是调用那个接口，本质就是会调用 Feign 创建的动态代理，这是核心中的核心
- Feign 的动态代理会根据你在接口上的@RequestMapping 等注解，来动态构造出你要请求的服务的地址
- 最后针对这个地址，发起请求、解析响应



微服务的容错处理（熔断） Hystrix

但是如果积分服务都挂了，每次调用都要去卡住几秒钟干啥呢？有意义吗？当然没有！所以我们直接对积分服务熔断不就得了，比如在 5 分钟内请求积分服务直接就返回了，不要去走网络请求卡住几秒钟，**这个过程，就是所谓的熔断！**

那人家又说，兄弟，积分服务挂了你就熔断，好歹你干点儿什么啊！别啥都不干就直接返回啊？没问题，咱们就来个降级：每次调用积分服务，你就在数据库里记录一条消息，说给某某用户增加了多少积分，因为积分服务挂了，导致没增加成功！这样等积分服务恢复了，你可以根据这些记录手工加一下积分。**这个过程，就是所谓的降级。**



微服务网关（智能路由） Zuul

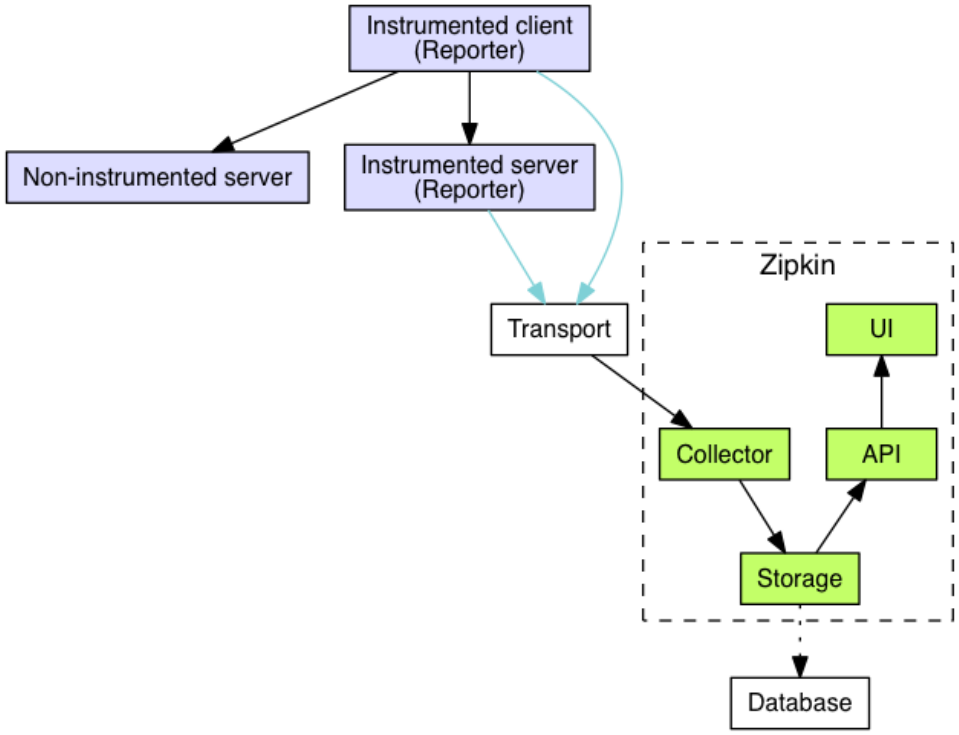
说完了 Hystrix，接着给大家说说最后一个组件：Zuul，也就是微服务网关。这个组件是负责网络路由的。

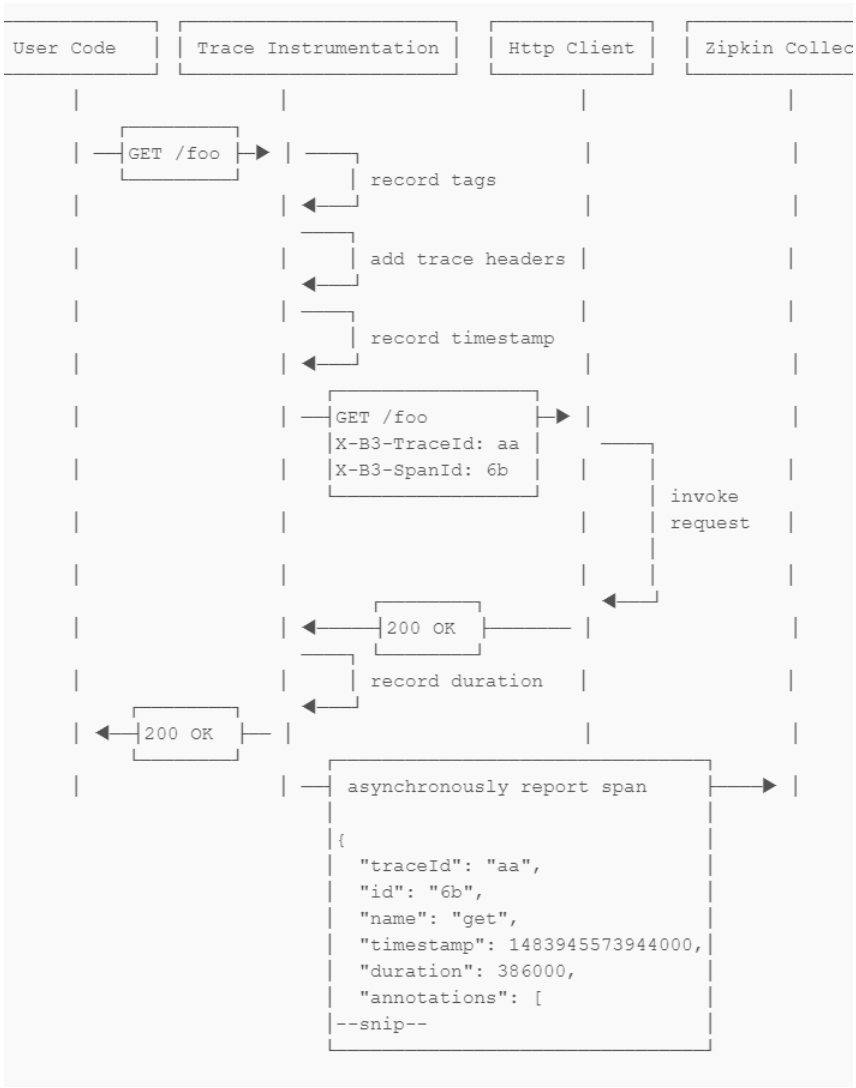
所以一般微服务架构中都必然会设计一个网关在里面，像 android、ios、pc 前端、微信小程序、H5 等等，不用去关心后端有几百个服务，就知道有一个网关，所有请求都往网关走，网关会根据请求中的一些特征，将请求转发给后端的各个服务。

而且有一个网关之后可以做**统一的降级、限流、认证授权、安全**，等等。

微服务跟踪 Sleuth

一般和 Zipkin 搭配使用




User Code
Trace Instrumentation
Http Client
Zipkin Collec
GET /foo
record tags
add trace headers
record timestamp
GET /foo
X-B3-TraceId: aa
X-B3-SpanId: 6b
invoke
request
200 OK
record duration
200 OK
asynchronously report span
{
"traceId": "aa",
"id": "6b",
"name": "get",
"timestamp": 1483945573944000,
"duration": 386000,
"annotations": [
--snip--