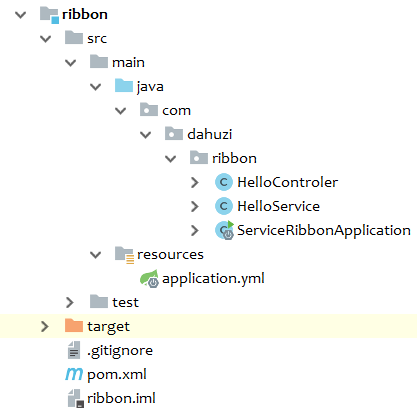
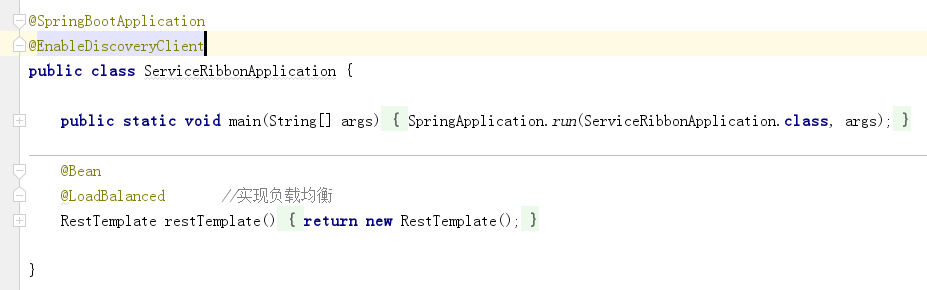
1. 服务的消费者的创建
2. 依赖的引入

<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-eureka</**artifactId**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-ribbon</**artifactId**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
</**dependency**>

1. 项目结构



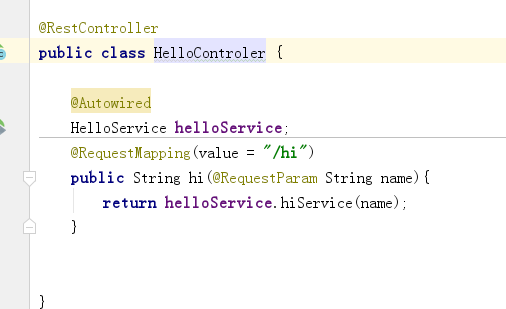
1. 工程搭建
2. 通过@EnableDiscoveryClient注解来指定该应用是eureka的客户端应用，并且获得服务发现的能力。创建RestTemplate（）实例，通过@LoadBalanced注解实现负载均衡



1. 创建HelloService类，并通过RestTemplate实例来调用前面创建的service-hi服务提供的”/hi”接口进行调用。在这这里我们发现，http地址中是根据设置的服务名来发现服务的，并不是IP地址；

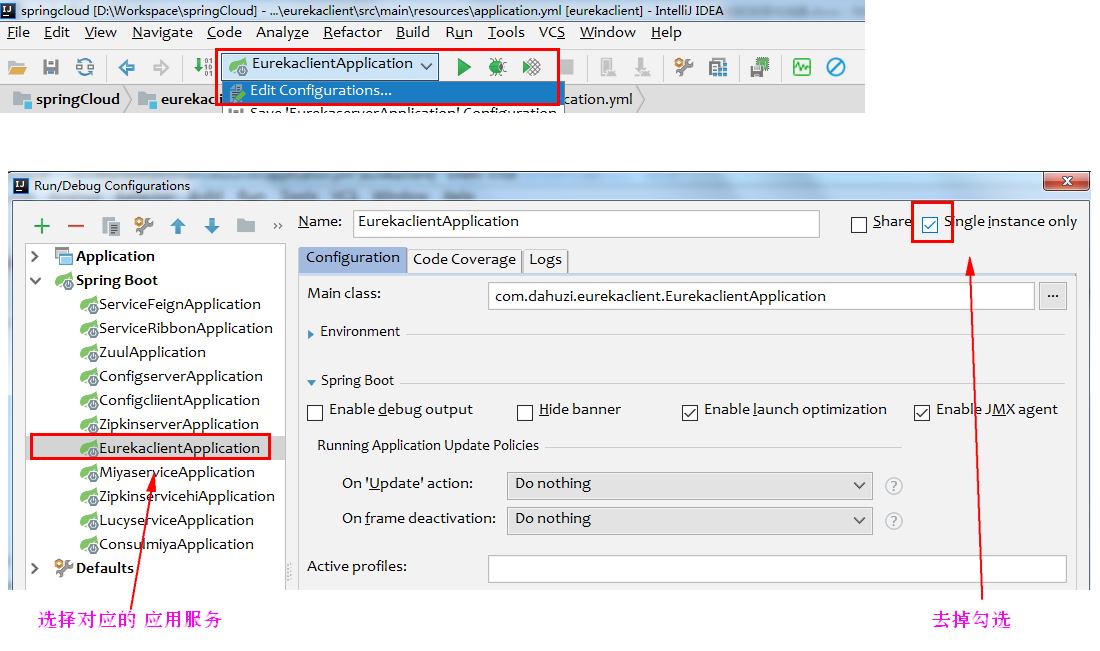


1. 创建一个controller，来调用service层

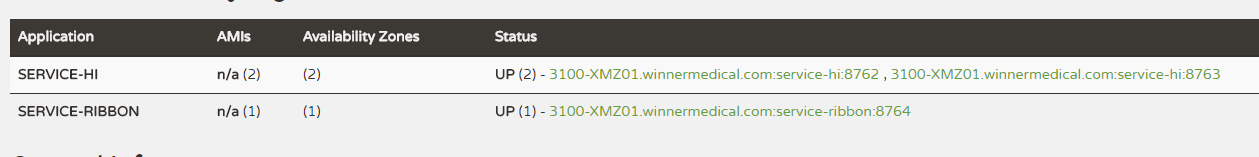


1. 启动项目
2. 启动两个eurekaClient服务，实现两个服务的提供者。端口号分别指定为8762、8763

PS：在idea中同一个项目指定不同的端口号分别启动方法：



1. 项目启动后



1. 3. 访问
2. <http://localhost:8764/hi?name=dahuzi （按住Ctrl点击）>
3. 总结：

从上面的案例我们可以看到，eureka的服务治理基础架构的三个核心要素是：

1.） 服务的注册中心：eurekaServer

2.） 服务的提供者： service-hi

3.） 服务的消费者：service-ribbon

1. 服务的提供者
2. 服务的注册

“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。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。

注意： eureka.cli ent.register-with-eureka的值不能设置为false，否则将不会在注册中心注册

1. 服务的同步

在eurekaServer集群的情况下，如果一个服务的提供者在像其中的一个eurekaServer注册中心注册以后，那么该注册中心就会把这个服务的提供者的信息转发到其他的注册中心去，从而实现服务的同步，那么服务的消费者也就可以从任意一台注册中心获取到服务提供者的信息

1. 服务的续约

服务的续约就是，在服务的提供者向注册中心注册服务以后呢，会用一个心跳（每隔一段时间）来告诉注册中心，我这个服务是可用的。目的是为了防止注册中心把服务剔除掉。

eureka.instance.lease-renewal-interval-in-seconds=30

eureka. instance.lease-expiration-duration-in-seconds=90

上面两个属性的设置分别控制的是：

第一个属性控制的是服务续约任务的调用的时间，默认的时间是30s；第二个属性控制的是用于定义服务失效的时间，默认为90秒。就是在注册中心90s没有收到心跳，就会认为服务是失效的。

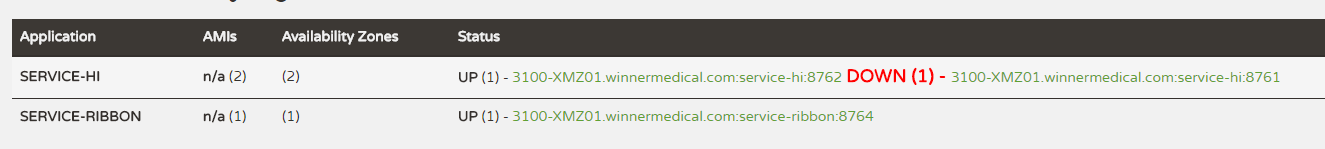
1. 服务的消费者
2. 获取服务

在服务的消费者启动的时候，会向注册中心发送一个请求，来获取注册中心服务注册清单，所以必须保证eureka. client.fetch-registry的值设置为true。因为这个属性是用来控制是否在注册中心检索服务的。注册中心为了性能达到最优，会每隔30秒来更新一次缓存中的服务注册清单。时间可以通过eureka.client.registry-fetch-interval-seconds属性来设置

1. 服务的调用

消费者会根据获取到的清单，和设置的调用的服务名选择对应的服务去调用。详细的说明在源码分析中说明

1. 服务的下线

当服务被正常关闭后，会向注册中心发送一个请求，告诉注册中心，已经关闭，此时注册中心会把服务的状态设置为（DOWN）,并把该服务的状态传播出去。如：

1. 服务注册
2. 失效剔除

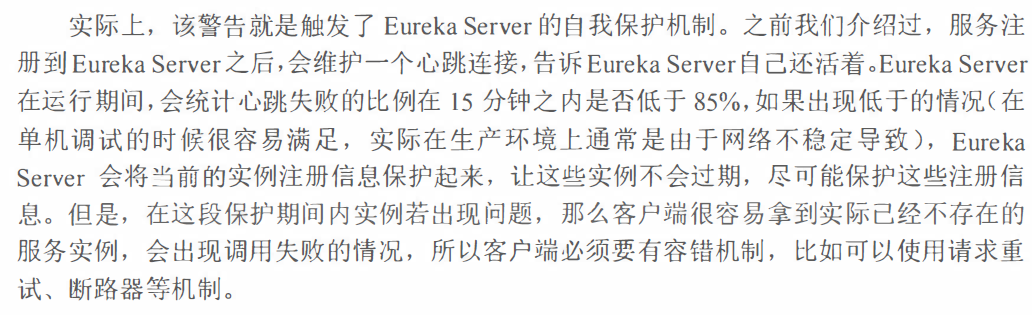
当eureka启动的时候，会创建一个定时任务，默认每隔60秒执行一次，把任务清单中超时的（默认90秒）没有续约注册的服务剔除出去

1. 自我保护

如下图：



触发eureka的自我保护机制的原因：



为了能让eurekaServer正确的剔除掉不可用的服务可以通过以下属性设置：

eureka.server. preservation=false（默认为true）

1. 实例测验

下面展示的实例测验的是在eurekaServer的自动保护关闭的状态下进行的。大致流程是，先启动eurekaServer，在分别启动service-hi，service-ribbon，查看eureka访问界面，停止service-hi，查看eureka访问界面

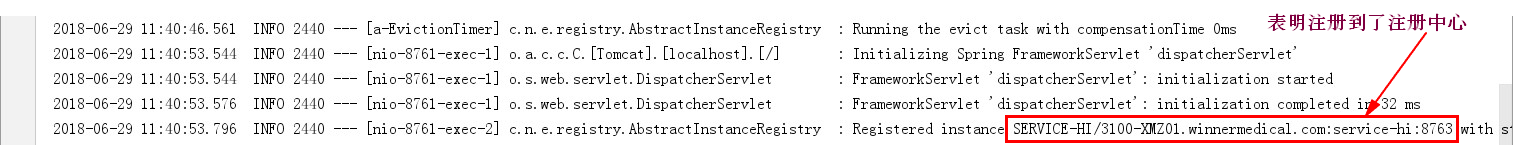
1. 启动eurekaServer



1. 启动service-hi



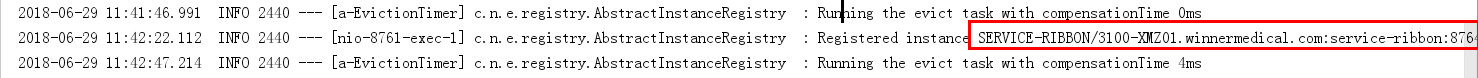
查看eurekaServer的控制台信息



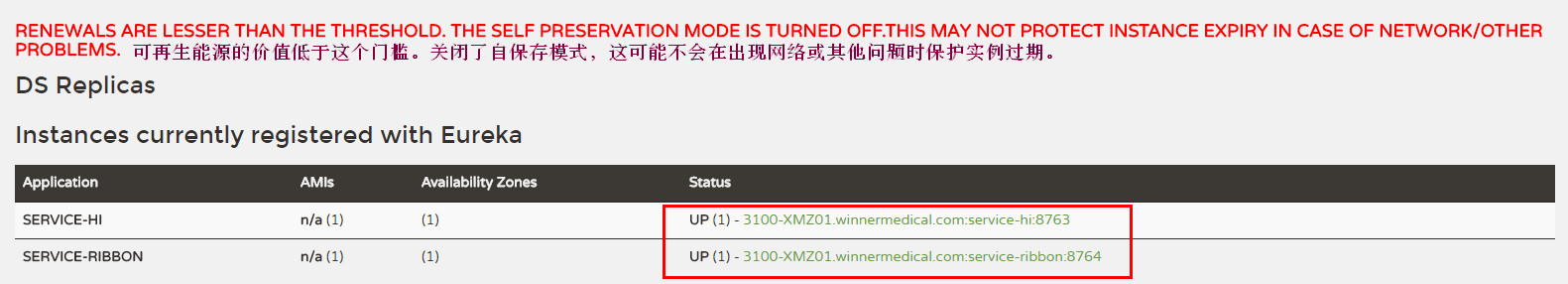
1. 启动service-ribbon



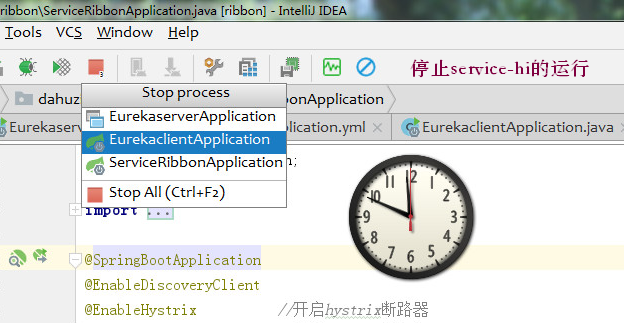
查看eurekaServer的控制台信息



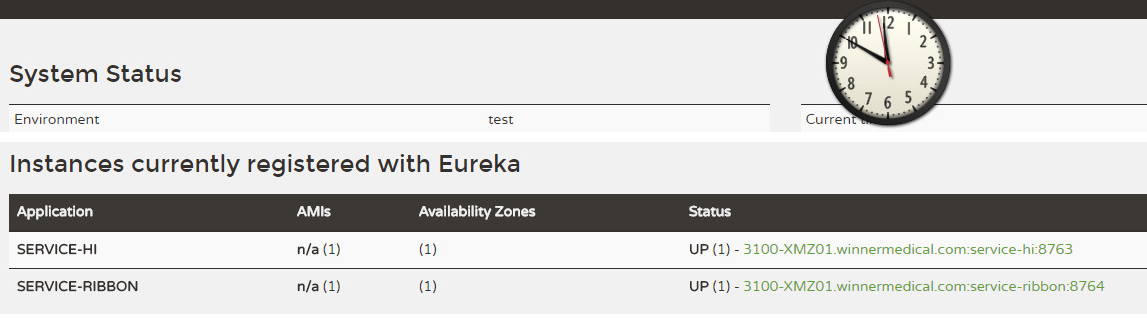
1. Eureka的主页（localHost：8761）



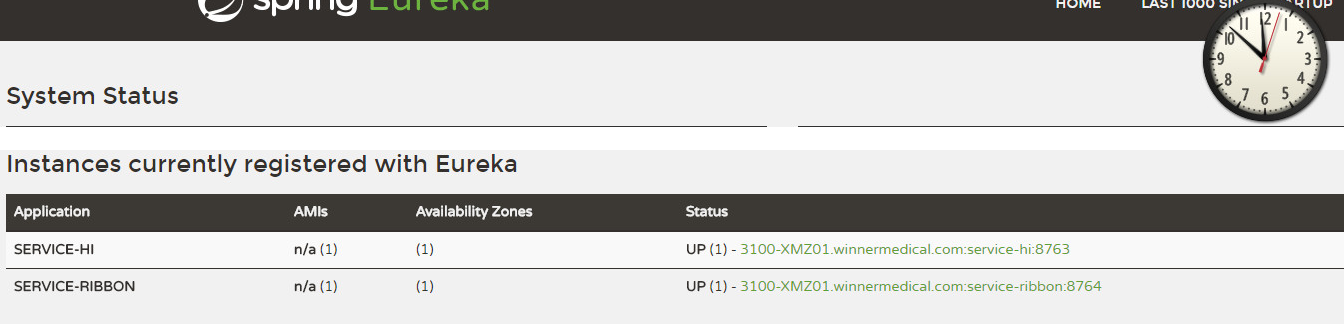
1. 停止service-hi的运行



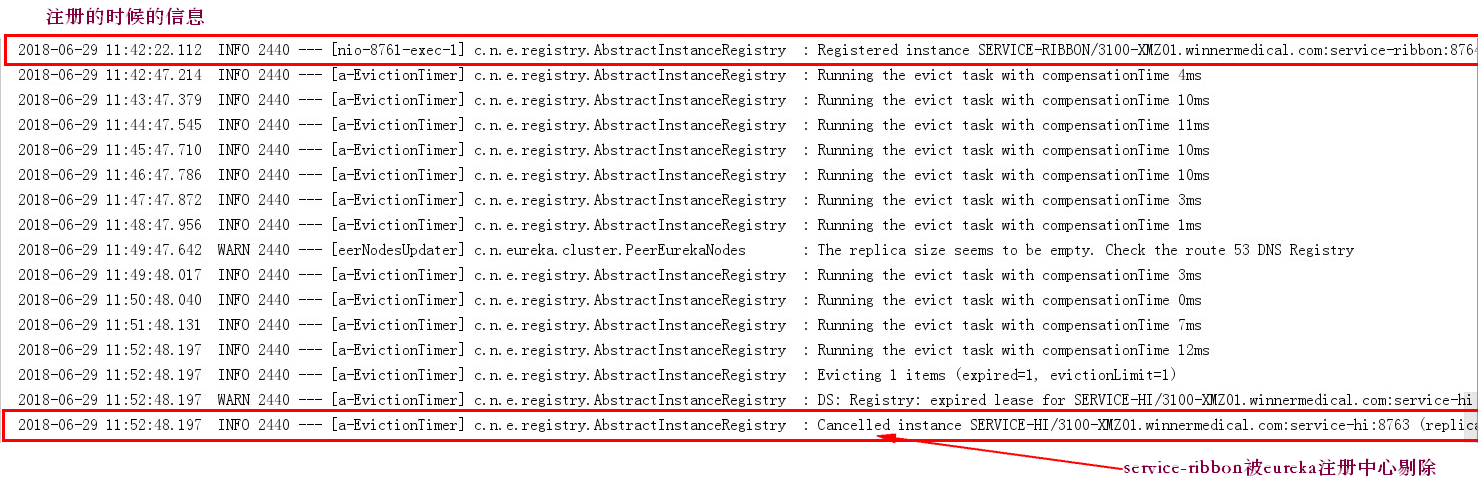
1. 在次刷新eureka的主页



真正停止service-hi到截图再刷新页面在截图，这中间会有一定的时间差，假设我们是在上图的时间停止的service-hi服务，那么下图的eureka主页就很能说明问题：



在两分钟左右的时间过去以后，eureka注册中心任然没有把关闭了的，没有了心跳的服务剔除出去，这说明什么？就下来在看看控制台打印了什么内容：



再次刷新eureka的主页

