**Eureka 两大组件**

Eureka Server：服务注册中心，用于提供服务注册功能。维护一个可用服务列表，存储了所有注册到注册中心的可用服务信息。

Eureka Client：客户端。通常指各个微服务。微服务启动后会自动向server发送心跳，默认30s。若Eureka Server在多个心跳周期没有收到某一个Eureka Client的心跳，Eureka Server将它从可用服务列表中移除，默认90s。

**Eureka中的元数据**

标准元数据：主机名、IP地址、端口号、状态页和健康检查等信息，这些都会被发布在服务注册表中，用于服务之间的调用。

自定义元数据：可以使用eureka.instance.metadata-map配置，符合key/value格式。这些数据可以在远程客户端中访问。

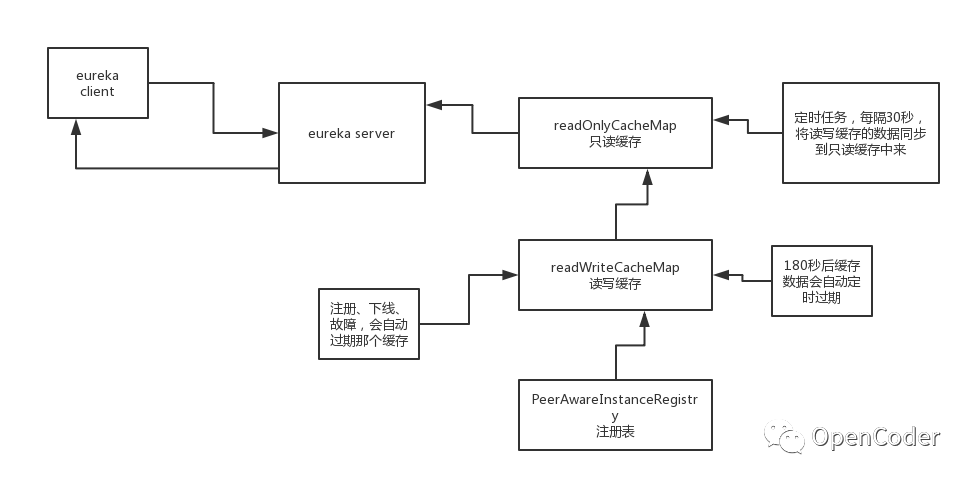
**Eureka 多重缓存机制**

核心方法：ApplicationsResource 类中的getContainers 方法。





先从readOnlyCacheMap中获取、后从readWriteCacheMap获取。最终从eureka server的注册表中读取。



**Eureka Server的注册表多级缓存的过期机制**

1. 主动过期

readWriteCacheMap 读写缓存，有新的服务实例发生注册、下线、故障的时候，会主动刷新readWriteCacheMap。调用ResponseCache.invalidate()。

将这个key对应的缓存给过期掉。

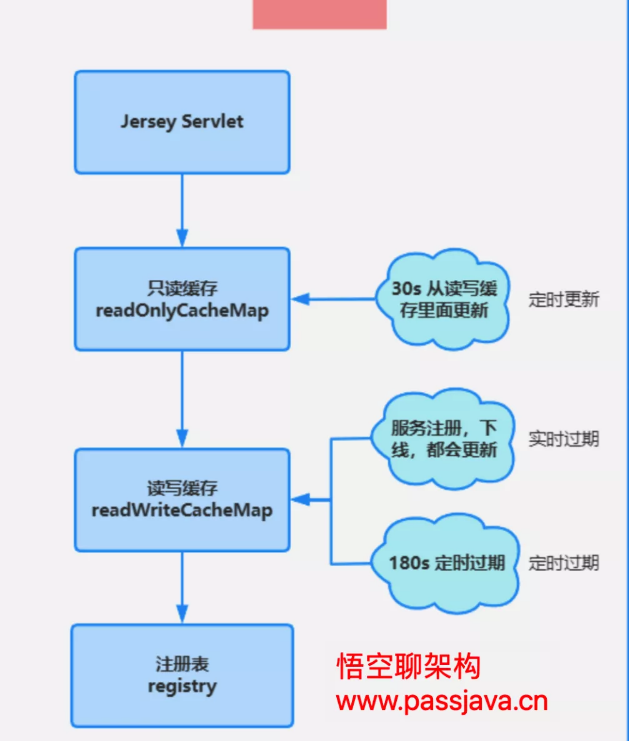
1. 定时过期

this.readWriteCacheMap = CacheBuilder.*newBuilder*().initialCapacity(serverConfig.getInitialCapacityOfResponseCache()).expireAfterWrite(serverConfig.getResponseCacheAutoExpirationInSeconds(), TimeUnit.*SECONDS*).removalListener(new RemovalListener<Key, ResponseCacheImpl.Value>() {  
 public void onRemoval(RemovalNotification<Key, ResponseCacheImpl.Value> notification) {  
 Key removedKey = (Key)notification.getKey();  
 if (removedKey.hasRegions()) {  
 Key cloneWithNoRegions = removedKey.cloneWithoutRegions();  
 ResponseCacheImpl.this.regionSpecificKeys.remove(cloneWithNoRegions, removedKey);  
 }  
  
 }  
})

构建readWriteCacheMap 构建的时候，可以指定自动过期时间。

1. 被动过期

默认每间隔30秒，执行一个定时调度的线程任务，TimerTask，对readWriteCacheMap 和readOnlyCacheMap中的数据进行对比，如果数据不一致，将readWriteCacheMap中的数据放置在readOnlyCacheMap中。



Eureka的自我保护机制

默认情况下，如果Eureka Server在90s未收到服务端心跳，Eureka Server会将该client移除。但是当网络出现故障时，微服务与Eureka Server之间无法通信，但微服务本身是正常运行的。此时不应该移除该微服务。因此，引入了【自我保护机制】。

使用自我保护机制能使Eureka 集群更加健壮、稳定。

