08 Eureka 缓存机制详细配置

更新时间：2019-06-19 17:54:55

> 立志是事业的大门，工作是登堂入室的旅程。
>
> ——巴斯德

上节为大家介绍了 Eureka 的工作原理，其中提到了 Eureka Server 内部有二层缓存机制，那这些机制是如何工作的，以及 Eureka Server 是如何存储服务的注册信息，本节会给大家揭晓。

Eureka 在使用过程中有一些非常重要的配置项，本节也会整理出来，方便大家以后在生产环境根据项目场景来调整。

Eureka Server 数据存储

我们知道 Eureka Server 在运行期间就是一个普通的 Java 项目，并没有使用数据库之类的存储软件，那么在运行期间是如何存储数据的呢？

Eureka Server 的数据存储分了两层：数据存储层和缓存层。数据存储层记录注册到 Eureka Server 上的服务信息，缓存层是经过包装后的数据，可以直接在 Eureka Client 调用时返回。我们先来看看数据存储层的数据结构。

Eureka Server 的数据存储层是双层的 ConcurrentHashMap，我们知道 ConcurrentHashMap 是线程安全高效的 Map 集合。

```
private final ConcurrentHashMap<String, Map<String, Lease<InstanceInfo>>> registry= new ConcurrentHashMap<String, Map<String, Lease<InstanceInfo>>>();
```

第一层的 ConcurrentHashMap 的 `key=spring.application.name`，也就是客户端实例注册的应用名；value 为嵌套的 ConcurrentHashMap。

第二层嵌套的 ConcurrentHashMap 的 key=instanceId，也就是服务的唯一实例 ID，value 为 Lease 对象，Lease 对象存储着这个实例的所有注册信息，包括 ip 、端口、属性等。

根据这个存储结构我们可以发现，Eureka Server 第一层都是存储着所有的服务名，以及服务名对应的实例信息，也就是说第一层都是按照服务应用名这个维度来切分存储：

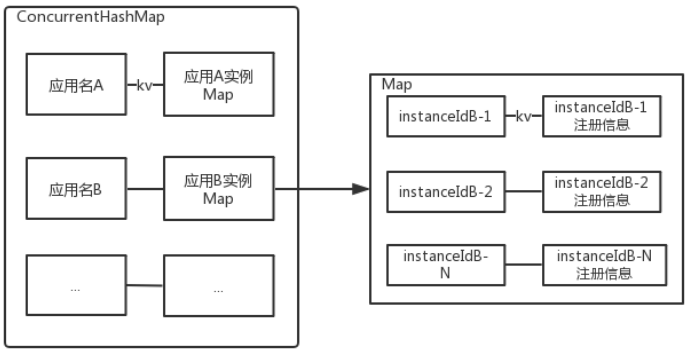
```
应用名1：应用1实例 Map
应该名2：应用2实例 Map
...
```

第二层是根据实例的唯一 ID 来存储的，那么按照这个结构最终的存储数据格式为：

```
          ：  应用1实例A：实例A的注册信息
应用名1：应用1实例：  应用1实例B：实例B的注册信息
          ：  应用1实例C：实例C的注册信息
          ：  ....
-----------------
          ：  应用2实例F：实例F的注册信息
应该名2：应用2实例：  应用2实例G：实例G的注册信息
          ：  ...

...
```

数据存储层数据结构如下图所示：



当如服务的状态发生变更时，会同步 Eureka Server 中的 registry 数据信息，比如服务注册、剔除服务时。

Eureka Server 缓存机制

Eureka Server 为了提供响应效率，提供了两层的缓存结构，将 Eureka Client 所需要的注册信息，直接存储在缓存结构中。

第一层缓存：readOnlyCacheMap，本质上是 ConcurrentHashMap，依赖定时从 readWriteCacheMap 同步数据，默认时间为 30 秒。

readOnlyCacheMap ： 是一个 CurrentHashMap 只读缓存，这个主要是为了供客户端获取注册信息时使用，其缓存更新，依赖于定时器的更新，通过和 readWriteCacheMap 的值做对比，如果数据不一致，则以 readWriteCacheMap 的数据为准。

第二层缓存：readWriteCacheMap，本质上是 Guava 缓存。

readWriteCacheMap：readWriteCacheMap 的数据主要同步于存储层。当获取缓存时判断缓存中是否没有数据，如果不存在此数据，则通过 CacheLoader 的 load 方法去加载，加载成功之后将数据放入缓存，同时返回数据。

readWriteCacheMap 缓存过期时间，默认为 180 秒，当服务下线、过期、注册、状态变更，都会来清除此缓存中的数据。

Eureka Client 获取全量或者增量的数据时，会先从一级缓存中获取；如果一级缓存中不存在，再从二级缓存中获取；如果二级缓存也不存在，这时候先将存储层的数据同步到缓存中，再从缓存中获取。

通过 Eureka Server 的二层缓存机制，可以非常有效地提升 Eureka Server 的响应时间，通过数据存储层和缓存层的数据切割，根据使用场景来提供不同的数据支持。

其它缓存设计

除过 Eureka Server 端存在缓存外，Eureka Client 也同样存在着缓存机制，Eureka Client 启动时会全量拉取服务列表，启动后每隔 30 秒从 Eureka Server 量获取服务列表信息，并保持在本地缓存中。

Eureka Client 增量拉取失败，或者增量拉取之后对比 hashcode 发现不一致，就会执行全量拉取，这样避免了网络某时段分片带来的问题，同样会更新到本地缓存。

同时对于服务调用，如果涉及到 ribbon 负载均衡，那么 ribbon 对于这个实例列表也有自己的缓存，这个缓存定时(默认30秒)从 Eureka Client 的缓存更新。

这么多的缓存机制可能就会造成一些问题，一个服务启动后可能最长需要 90s 才能被其它服务感知到：

1、首先，Eureka Server 维护每 30s 更新的响应缓存

2、Eureka Client 对已经获取到的注册信息也做了 30s 缓存

3、负载均衡组件 Ribbon 也有 30s 缓存

这三个缓存加起来，就有可能导致服务注册最长延迟 90s ，这个需要我们在特殊业务场景中注意其产生的影响。

Eureka 常用配置

在 Spring Cloud 官网查看 Spring Cloud Netflix 的介绍时，会出现以下几个名字，在这里给大家一一介绍：

- Eureka Server，Eureka 作为注册中心的角色，称之为 Eureka Server。
- Eureka Client，这个比较好理解，前面的文章介绍过注册到 Eureka 的客户端。
- Eureka Instance，可以称之为 Eureka 实例，注册到服务中心的每一个单独的示例，比如 ip 为 192.168.0.1 、端口为 8080 的一个实例。
- Eureka service，Eureka service 其实是 Eureka Instance 的一个抽象，比如是订单服务，有可能订单服务是由多个订单实例（Eureka Instance）组成，具体表现为配置应用名相同的一组实例。

接下来我们就来了解上面这几个概念的常用配置项，Eureka service 作为抽象的概念没有具体的配置项，其它三个均可配置。

Eureka Server 常用配置

```
#服务端开启自我保护模式，前面章节有介绍
eureka.server.enable-self-preservation=true
#扫描失效服务的间隔时间（单位毫秒，默认是60*1000）即60秒
eureka.server.eviction-interval-timer-in-ms= 60000
#间隔多长时间，清除过期的 delta 数据
eureka.server.delta-retention-timer-interval-in-ms=0
#请求频率限制器
eureka.server.rate-limiter-burst-size=10
#是否开启请求频率限制器
eureka.server.rate-limiter-enabled=false
```

```
#请求频率的平均值
eureka.server.rate-limiter-full-fetch-average-rate=100
#是否对标准的client进行频率请求限制。如果是false，则只对非标准client进行限制
eureka.server.rate-limiter-throttle-standard-clients=false
#注册服务、拉去服务列表数据的请求频率的平均值
eureka.server.rate-limiter-registry-fetch-average-rate=500
#设置信任的client list
eureka.server.rate-limiter-privileged-clients=
#在设置的时间范围类，期望与client续约的百分比。
eureka.server.renewal-percent-threshold=0.85
#多长时间更新续约的阈值
eureka.server.renewal-threshold-update-interval-ms=0
#对于缓存的注册数据，多长时间过期
eureka.server.response-cache-auto-expiration-in-seconds=180
#多长时间更新一次缓存中的服务注册数据
eureka.server.response-cache-update-interval-ms=0
#缓存增量数据的时间，以便在检索的时候不丢失信息
eureka.server.retention-time-in-m-s-in-delta-queue=0
#当时间戳不一致的时候，是否进行同步
eureka.server.sync-when-timestamp-differs=true
#是否采用只读缓存策略，只读策略对于缓存的数据不会过期。
eureka.server.use-read-only-response-cache=true


#################server node 与 node 之间关联的配置#####################33
#发送复制数据是否在request中，总是压缩
eureka.server.enable-replicated-request-compression=false
#指示群集节点之间的复制是否应批处理以提高网络效率。
eureka.server.batch-replication=false
#允许备份到备份池的最大复制事件数量。而这个备份池负责除状态更新的其他事件。可以根据内存大小，超时和复制流量，来设置此值得大小
eureka.server.max-elements-in-peer-replication-pool=10000
#允许备份到状态备份池的最大复制事件数量
eureka.server.max-elements-in-status-replication-pool=10000
#多个服务中心相互同步信息线程的最大空闲时间
eureka.server.max-idle-thread-age-in-minutes-for-peer-replication=15
#状态同步线程的最大空闲时间
eureka.server.max-idle-thread-in-minutes-age-for-status-replication=15
#服务注册中心各个instance相互复制数据的最大线程数量
eureka.server.max-threads-for-peer-replication=20
#服务注册中心各个instance相互复制状态数据的最大线程数量
eureka.server.max-threads-for-status-replication=1
#instance之间复制数据的通信时长
eureka.server.max-time-for-replication=30000
#正常的对等服务instance最小数量。-1表示服务中心为单节点。
eureka.server.min-available-instances-for-peer-replication=-1
#instance之间相互复制开启的最小线程数量
eureka.server.min-threads-for-peer-replication=5
#instance之间用于状态复制，开启的最小线程数量
eureka.server.min-threads-for-status-replication=1
#instance之间复制数据时可以重试的次数
eureka.server.number-of-replication-retries=5
#eureka节点间间隔多长时间更新一次数据。默认10分钟。
eureka.server.peer-eureka-nodes-update-interval-ms=600000
#eureka服务状态的相互更新的时间间隔。
eureka.server.peer-eureka-status-refresh-time-interval-ms=0
#eureka对等节点间连接超时时间
eureka.server.peer-node-connect-timeout-ms=200
#eureka对等节点连接后的空闲时间
eureka.server.peer-node-connection-idle-timeout-seconds=30
#节点间的读数据连接超时时间
eureka.server.peer-node-read-timeout-ms=200
#eureka server 节点间连接的总共最大数量
eureka.server.peer-node-total-connections=1000
#eureka server 节点间连接的单机最大数量
eureka.server.peer-node-total-connections-per-host=10
#在服务节点启动时，eureka尝试获取注册信息的次数
eureka.server.registry-sync-retries=
#在服务节点启动时，eureka多次尝试获取注册信息的间隔时间
eureka.server.registry-sync-retry-wait-ms=
#当eureka server启动的时候，不能从对等节点获取instance注册信息的情况，应等待多长时间。
eureka.server.wait-time-in-ms-when-sync-empty=0
```

Eureka Client 常用配置

```
#该客户端是否可用
eureka.client.enabled=true
#实例是否在eureka服务器上注册自己的信息以供其他服务发现，默认为true
eureka.client.register-with-eureka=false
#此客户端是否获取eureka服务器注册表上的注册信息，默认为true
eureka.client.fetch-registry=false
#是否过滤掉，非UP的实例。默认为true
eureka.client.filter-only-up-instances=true
#与Eureka注册服务中心的通信zone和url地址
eureka.client.serviceUrl.defaultZone=http://${eureka.instance.hostname}:${server.port}/eureka/

#client连接Eureka服务端后的空闲等待时间，默认为30 秒
eureka.client.eureka-connection-idle-timeout-seconds=30
#client连接eureka服务端的连接超时时间，默认为5秒
eureka.client.eureka-server-connect-timeout-seconds=5
#client对服务端的读超时时长
eureka.client.eureka-server-read-timeout-seconds=8
#client连接all eureka服务端的总连接数，默认200
eureka.client.eureka-server-total-connections=200
#client连接eureka服务端的单机连接数量，默认50
eureka.client.eureka-server-total-connections-per-host=50
#执行程序指数回退刷新的相关属性，是重试延迟的最大倍数值，默认为10
eureka.client.cache-refresh-executor-exponential-back-off-bound=10
#执行程序缓存刷新线程池的大小，默认为5
eureka.client.cache-refresh-executor-thread-pool-size=2
#心跳执行程序回退相关的属性，是重试延迟的最大倍数值，默认为10
eureka.client.heartbeat-executor-exponential-back-off-bound=10
#心跳执行程序线程池的大小,默认为5
eureka.client.heartbeat-executor-thread-pool-size=5
# 询问Eureka服务url信息变化的频率（s），默认为300秒
eureka.client.eureka-service-url-poll-interval-seconds=300
#最初复制实例信息到eureka服务器所需的时间（s），默认为40秒
eureka.client.initial-instance-info-replication-interval-seconds=40
#间隔多长时间再次复制实例信息到eureka服务器，默认为30秒
eureka.client.instance-info-replication-interval-seconds=30
#从eureka服务器注册表中获取注册信息的时间间隔（s），默认为30秒
eureka.client.registry-fetch-interval-seconds=30

# 获取实例所在的地区。默认为us-east-1
eureka.client.region=us-east-1
#实例是否使用同一zone里的eureka服务器，默认为true，理想状态下，eureka客户端与服务端是在同一zone下
eureka.client.prefer-same-zone-eureka=true
# 获取实例所在的地区下可用性的区域列表，用逗号隔开。（AWS）
eureka.client.availability-zones.china=defaultZone,defaultZone1,defaultZone2
#eureka服务注册表信息里的以逗号隔开的地区名单，如果不这样返回这些地区名单，则客户端启动将会出错。默认为null
eureka.client.fetch-remote-regions-registry=
#服务器是否能够重定向客户端请求到备份服务器。 如果设置为false，服务器将直接处理请求，如果设置为true，它可能发送HTTP重定向到客户端。
默认为false
eureka.client.allow-redirects=false
#客户端数据接收
eureka.client.client-data-accept=
#增量信息是否可以提供给客户端看，默认为false
eureka.client.disable-delta=false
#eureka服务器序列化/反序列化的信息中获取“_”符号的的替换字符串。默认为“__“
eureka.client.escape-char-replacement=__
#eureka服务器序列化/反序列化的信息中获取“$”符号的替换字符串。默认为“_-”
eureka.client.dollar-replacement="_-"
#当服务端支持压缩的情况下，是否支持从服务端获取的信息进行压缩。默认为true
eureka.client.g-zip-content=true
#是否记录eureka服务器和客户端之间在注册表的信息方面的差异，默认为false
eureka.client.log-delta-diff=false
# 如果设置为true,客户端的状态更新将会点播更新到远程服务器上，默认为true
eureka.client.on-demand-update-status-change=true
#此客户端只对一个单一的VIP注册表的信息感兴趣。默认为null
eureka.client.registry-refresh-single-vip-address=
#client是否在初始化阶段强行注册到服务中心，默认为false
eureka.client.should-enforce-registration-at-init=false
#client在shutdown的时候是否显示的注销服务从服务中心，默认为true
eureka.client.should-unregister-on-shutdown=true
```

Eureka Instance 常用配置

```
#服务注册中心实例的主机名
eureka.instance.hostname=localhost
#注册在Eureka服务中的应用组名
eureka.instance.app-group-name=
#注册在的Eureka服务中的应用名称
eureka.instance.appname=
#该实例注册到服务中心的唯一ID
eureka.instance.instance-id=
#该实例的IP地址
eureka.instance.ip-address=
#该实例，相较于hostname是否优先使用IP
eureka.instance.prefer-ip-address=false
```

小结

本节课给大家介绍了 Eureka 数据的存储原理，以及内部巧妙的二级缓存机制，在我们学习 Eureka 的内部存储原理时，同时了解到 Eureka 内部的精妙设计，后期我们在项目中也可借鉴它的实现机制。

本节的最后也给大家罗列了一些 Eureka 的常用配置，方便大家在日常工作中根据需要来调整。

参考出处：

https://www.infoq.cn/article/jlDJQ*3wtN2PcqTDyokh

https://blog.51cto.com/881206524/2117014

本文作者：纯洁的微笑、江南一点雨

07 Eureka 工作原理

09 Consul 注册中心介绍