## Spring Cloud限流详解（附源码）

原创 2017-12-01 周立 [A周立SpringCloud](http://mp.weixin.qq.com/s/mh4BFWOsXakrn17XvahszA##)

在高并发的应用中，**限流**往往是一个绕不开的话题。本文详细探讨在Spring Cloud中如何实现限流。

在 Zuul 上实现限流是个不错的选择，只需要编写一个过滤器就可以了，关键在于如何实现限流的算法。常见的限流算法有漏桶算法以及令牌桶算法。这个可参考 https://www.cnblogs.com/LBSer/p/4083131.html ，写得通俗易懂，你值得拥有，我就不拽文了。

GoogleGuava 为我们提供了限流工具类 RateLimiter ，于是乎，我们可以撸代码了。

## 代码示例

1. @Component
2. public class RateLimitZuulFilter extends ZuulFilter {
3. private final RateLimiter rateLimiter = RateLimiter.create(1000.0);
4. @Override
5. public String filterType() {
6. return FilterConstants.PRE\_TYPE;
7. }
8. @Override
9. public int filterOrder() {
10. return Ordered.HIGHEST\_PRECEDENCE;
11. }
12. @Override
13. public boolean shouldFilter() {
14. // 这里可以考虑弄个限流开启的开关，开启限流返回true，关闭限流返回false，你懂的。
15. return true;
16. }
17. @Override
18. public Object run() {
19. try {
20. RequestContext currentContext = RequestContext.getCurrentContext();
21. HttpServletResponse response = currentContext.getResponse();
22. if (!rateLimiter.tryAcquire()) {
23. HttpStatus httpStatus = HttpStatus.TOO\_MANY\_REQUESTS;
24. response.setContentType(MediaType.TEXT\_PLAIN\_VALUE);
25. response.setStatus(httpStatus.value());
26. response.getWriter().append(httpStatus.getReasonPhrase());
27. currentContext.setSendZuulResponse(false);
28. throw new ZuulException(
29. httpStatus.getReasonPhrase(),
30. httpStatus.value(),
31. httpStatus.getReasonPhrase()
32. );
33. }
34. } catch (Exception e) {
35. ReflectionUtils.rethrowRuntimeException(e);
36. }
37. return null;
38. }
39. }

如上，我们编写了一个 pre 类型的过滤器。对Zuul过滤器有疑问的可参考我的博客：

* Spring Cloud内置的Zuul过滤器详解：http://www.itmuch.com/spring-cloud/zuul/zuul-filter-in-spring-cloud
* Spring Cloud Zuul过滤器详解：http://www.itmuch.com/spring-cloud/zuul/spring-cloud-zuul-filter

在过滤器中，我们使用 GuavaRateLimiter 实现限流，如果已经达到最大流量，就抛异常。

## 分布式场景下的限流

以上单节点Zuul下的限流，但在生产中，我们往往会有多个Zuul实例。对于这种场景如何限流呢？我们可以借助Redis实现限流。

使用redis实现，存储两个key，一个用于计时，一个用于计数。请求每调用一次，计数器增加1，若在计时器时间内计数器未超过阈值，则可以处理任务

1. if(!cacheDao.hasKey(TIME\_KEY)) {
2. cacheDao.putToValue(TIME\_KEY, 0, 1, TimeUnit.SECONDS);
3. }
4. if(cacheDao.hasKey(TIME\_KEY) && cacheDao.incrBy(COUNTER\_KEY, 1) > 400) {
5. // 抛个异常什么的
6. }

## 实现微服务级别的限流

一些场景下，我们可能还需要实现微服务粒度的限流。此时可以有两种方案：

**方式一：在微服务本身实现限流。**

和在Zuul上实现限流类似，只需编写一个过滤器或者拦截器即可，比较简单，不作赘述。个人不太喜欢这种方式，因为每个微服务都得编码，感觉成本很高啊。

加班那么多，作为程序猿的我们，应该学会偷懒，这样才可能有时间孝顺父母、抱老婆、逗儿子、遛狗养鸟、聊天打屁、追求人生信仰。好了不扯淡了，看方法二吧。

**方法二：在Zuul上实现微服务粒度的限流。**

在讲解之前，我们不妨模拟两个路由规则，两种路由规则分别代表Zuul的两种路由方式。

1. zuul:
2. routes:
3. microservice-provider-user: /user/\*\*
4. user2:
5. url: http://localhost:8000/
6. path: /user2/\*\*

如配置所示，在这里，我们定义了两个路由规则， microservice-provider-user 以及 user2 ，其中 microservice-provider-user 这个路由规则使用到Ribbon + Hystrix，走的是 RibbonRoutingFilter ；而 user2 这个路由用不上Ribbon也用不上Hystrix，走的是 SipleRoutingFilter 。如果你搞不清楚这点，请参阅我的博客：

* Spring Cloud内置的Zuul过滤器详解：http://www.itmuch.com/spring-cloud/zuul/zuul-filter-in-spring-cloud
* Spring Cloud Zuul过滤器详解：http://www.itmuch.com/spring-cloud/zuul/spring-cloud-zuul-filter

搞清楚这点之后，我们就可以撸代码了：

1. @Component
2. public class RateLimitZuulFilter extends ZuulFilter {
3. private Map<String, RateLimiter> map = Maps.newConcurrentMap();
4. @Override
5. public String filterType() {
6. return FilterConstants.PRE\_TYPE;
7. }
8. @Override
9. public int filterOrder() {
10. // 这边的order一定要大于org.springframework.cloud.netflix.zuul.filters.pre.PreDecorationFilter的order
11. // 也就是要大于5
12. // 否则，RequestContext.getCurrentContext()里拿不到serviceId等数据。
13. return Ordered.LOWEST\_PRECEDENCE;
14. }
15. @Override
16. public boolean shouldFilter() {
17. // 这里可以考虑弄个限流开启的开关，开启限流返回true，关闭限流返回false，你懂的。
18. return true;
19. }
20. @Override
21. public Object run() {
22. try {
23. RequestContext context = RequestContext.getCurrentContext();
24. HttpServletResponse response = context.getResponse();
25. String key = null;
26. // 对于service格式的路由，走RibbonRoutingFilter
27. String serviceId = (String) context.get(SERVICE\_ID\_KEY);
28. if (serviceId != null) {
29. key = serviceId;
30. map.putIfAbsent(serviceId, RateLimiter.create(1000.0));
31. }
32. // 如果压根不走RibbonRoutingFilter，则认为是URL格式的路由
33. else {
34. // 对于URL格式的路由，走SimpleHostRoutingFilter
35. URL routeHost = context.getRouteHost();
36. if (routeHost != null) {
37. String url = routeHost.toString();
38. key = url;
39. map.putIfAbsent(url, RateLimiter.create(2000.0));
40. }
41. }
42. RateLimiter rateLimiter = map.get(key);
43. if (!rateLimiter.tryAcquire()) {
44. HttpStatus httpStatus = HttpStatus.TOO\_MANY\_REQUESTS;
45. response.setContentType(MediaType.TEXT\_PLAIN\_VALUE);
46. response.setStatus(httpStatus.value());
47. response.getWriter().append(httpStatus.getReasonPhrase());
48. context.setSendZuulResponse(false);
49. throw new ZuulException(
50. httpStatus.getReasonPhrase(),
51. httpStatus.value(),
52. httpStatus.getReasonPhrase()
53. );
54. }
55. } catch (Exception e) {
56. ReflectionUtils.rethrowRuntimeException(e);
57. }
58. return null;
59. }
60. }

简单讲解一下这段代码：

对于 microservice-provider-user 这个路由，我们可以用 context.get(SERVICE\_ID\_KEY); 获取到serviceId，获取出来就是 microservice-provider-user；

而对于 user2 这个路由，我们使用 context.get(SERVICE\_ID\_KEY); 获得是null，但是呢，可以用 context.getRouteHost() 获得路由到的地址，获取出来就是 http://localhost:8000/ 。接下来的事情，你们懂的。

## 改进与提升

实际项目中，除以上实现的限流方式，还可能会：

**一、在上文的基础上，增加配置项，控制每个路由的限流指标，并实现动态刷新**，从而实现更加灵活的管理

**二、基于CPU、内存、数据库等压力限流（感谢平安常浩智）提出。**。

下面，笔者借助Spring Boot Actuator提供的 Metrics 能力进行实现基于内存压力的限流——当可用内存低于某个阈值就开启限流，否则不开启限流。

1. @Component
2. public class RateLimitZuulFilter extends ZuulFilter {
3. @Autowired
4. private SystemPublicMetrics systemPublicMetrics;
5. @Override
6. public boolean shouldFilter() {
7. // 这里可以考虑弄个限流开启的开关，开启限流返回true，关闭限流返回false，你懂的。
8. Collection<Metric<?>> metrics = systemPublicMetrics.metrics();
9. Optional<Metric<?>> freeMemoryMetric = metrics.stream()
10. .filter(t -> "mem.free".equals(t.getName()))
11. .findFirst();
12. // 如果不存在这个指标，稳妥起见，返回true，开启限流
13. if (!freeMemoryMetric.isPresent()) {
14. return true;
15. }
16. long freeMemory = freeMemoryMetric.get()
17. .getValue()
18. .longValue();
19. // 如果可用内存小于1000000KB，开启流控
20. return freeMemory < 1000000L;
21. }
22. // 省略其他方法
23. }

**三、实现不同维度的限流，**例如：

* 对请求的目标URL进行限流（例如：某个URL每分钟只允许调用多少次）
* 对客户端的访问IP进行限流（例如：某个IP每分钟只允许请求多少次）
* 对某些特定用户或者用户组进行限流（例如：非VIP用户限制每分钟只允许调用100次某个API等）
* 多维度混合的限流。此时，就需要实现一些限流规则的编排机制。与、或、非等关系。

## 参考文档

* 分布式环境下限流方案的实现：http://blog.csdn.net/Justnow\_/article/details/53055299