如何分流一万个领鸡蛋的老头——平滑加权轮询算法实战

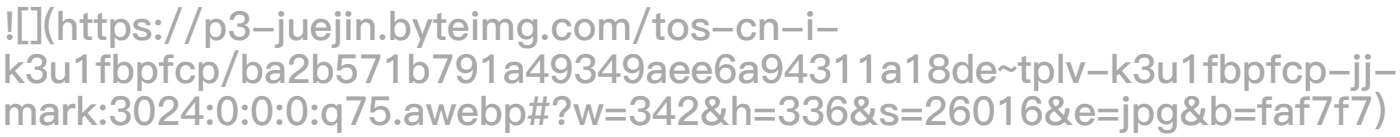
小剧场

经过五一结束的调休，小剧场的主人公程序员老马正好结束了6天连班的痛苦，正享受着周末的懒觉。正在这清净美好的时刻，只听见楼下仿佛有万马奔腾，老马揉着惺忪睡眼开窗一看，原来是楼下的超市新开张，有一万个老头在楼下正等着领鸡蛋呢

老马看着这一眼望不到头的队伍顿感两眼一黑，这等他们领完鸡蛋，我这假期也得完了，我倒要来看看怎么个事，可不能让这群老头毁了这来之不易的假期

怎么个事

老马好不容易挤到队伍的最前面，看着负责人正满头大汗的分配着柜台大妈给老头填会员表呢。众所周知，这超市的鸡蛋可不是白领的，想领鸡蛋还得先填上你的个人信息，方便他们后续跟你 发垃圾信息 联络感情



老马：你们这领个鸡蛋怎么这么费劲呢？你遇到啥问题了跟我说说

负责人：你有所不知，这领鸡蛋之前得先登记会员。这可都是办理业务的大妈的业绩。她们有的人手脚麻利头脑清晰，1个人的效率顶的上3个人，给这些效率不同的大妈分配业务可难咯。得事先定好他们接待业务的比例，多了少了都会有意见，效率高的人业务多，但是也不能让人连着连着干，一人干活，其他人干看着也不行。

老马：看来小伙子你是没有好好学过负载均衡算法啊，你看我略微出手

需求分析

首先给大妈们按照工作能力计算一下权重，能力强的效率高权重就大

```
广场舞大妈:权重3 普通大妈:权重2 老花眼大妈:权重1
```

又经过了一阵讨论，负责人说他期望把业务按照这样的顺序分配下去

```
广场舞大妈 -> 普通大妈 -> 老花眼大妈 -> 广场舞大妈 -> 普通大妈 -> 广场舞大妈
```

想实现这样的效果，在负载均衡的业务中早已有了解决方案，那就是平滑加权轮询算法

什么是平滑加权轮询算法

平滑加权轮询算法是一种巧妙的动态权重值算法，它最早出现在网络负载均衡中，在普通的加权算法基础之上额外定义了动态权重值。

如果我们用之前的大妈举例，那么如下所示，他们的初始权重值都是0

	唯一标识 id	权重 weight	动态权重 current
广场舞大妈	A	3	0
普通大妈	B	2	0
老花眼大妈	C	1	0

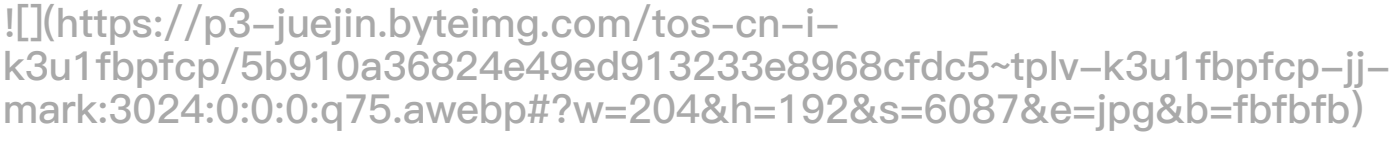
每次有业务过来，先计算动态权重，每个人的动态权重+自己的权重，然后取动态权重最高的，之后其他人不变，动态权重最高的需要减去 权重之和，那么我们看一下执行6次的之后她们的动态权重值的变化

次数		A	B	C	动态权重最高者
1	执行前	0	0	0	
1	加自身权重后	3	2	1	A
1	最高者减权重和	-3	2	1	
2	执行前	-3	2	1	
2	加自身权重后	0	4	2	B
2	最高者减权重和	0	-2	2	
3	执行前	0	-2	2	
3	加自身权重后	3	0	3	C
3	最高者减权重和	3	0	-3	
4	执行前	3	0	-3	
4	加自身权重后	6	2	-2	A
4	最高者减权重和	0	2	-2	
5	执行前	0	2	-2	
5	加自身权重后	3	4	-1	B
5	最高者减权重和	3	-2	-1	
6	执行前	3	-2	-1	
6	加自身权重后	6	0	0	A
6	最高者减权重和	0	0	0	

果然和预期一致，输出次数与权重吻合，并且相互分割，并没有出现权重高的连续的情况

实战之前

验证了算法可行，实际开发之前我又有了几个点子



这个算法每次请求时都需要遍历一遍所有节点

我们通过观察可知，在循环 `s=权重比例之和` 次后完成一次循环

也就是说在权重比例不变的情况下，每次循环取出的id顺序时相同的

那么我只需要计算一次，然后报存一下这个队列，每次按这个队列的顺序取出id就行了

我们可以记录请求次数，每次取队列的下标 `请求次数%队列长度`

代码实战

这个部分如果不写Java的可以略过~

首先定义一些配置，这里我选择用redis缓存数据

并且做一些自定义配置 以防出现rediskey重复或者数据过大

```
server:
  port: 8770
spring:
  redis:
    host: localhost
    port: 6379
    database: 0
# 自定义配置
swrb:
  redis-prefix: "swrb:"
  node-list-max-size: 99
  node-max-weight: 99
```

使用 `@ConfigurationProperties` 注解可以很方便的一次性读取指定前缀下的所有配置

在Spring环境中可以使用 `@Component` 或者 `@EnableConfigurationProperties(SwrbProperties.class)` 把这个类注入到Spring中

```
@ConfigurationProperties(prefix = ”swrb”)
@Data
public class SwrbProperties {

    /**
     * redis前缀
     */
    private String redisPrefix = ”swrb:”;

    /**
     * 节点列表最大长度
     */
    private int nodeListMaxSize = 100;

    /**
     * 节点最大权重
     */
    private int nodeMaxWeight = 100;

}
```

先定义一下节点类，每次先设置好所有的节点

```
@NoArgsConstructor
@AllArgsConstructor
@Data
public class Node {
    private String id;
    private Integer weight;
    private Integer current;

    public static Node create(String id, Integer weight) {
        return new Node(id, weight, 0);
    }
}
```

然后根据前面的算法计算 ,调用 calculateIdList(nodeList)

```
public class NodeCalculateUtil {

    public static List<String> calculateIdList(List<Node> nodeList) {
        Assert.notEmpty(nodeList);
        if (nodeList.size() == 1) {
            return
nodeList.stream().map(Node::getId).collect(Collectors.toList());
        }
        List<Integer> weightList =
nodeList.stream().map(Node::getWeight).collect(Collectors.toList());

        int totalWeight = nodeList.stream().map(Node::getWeight).reduce(0,
Integer::sum);
        // 计算一轮循环需要的最小次数
        int gcd = gcdOfArray(weightList);
        int roundTimes = totalWeight / gcd;

        List<String> idList = new ArrayList<>(roundTimes);

        // 复制 nodeList
        List<Node> copyNodeList = CollUtil.newArrayList(nodeList);

        for (int i = 0; i < roundTimes; i++) {
            String nodeId = calculateNodeId(copyNodeList, totalWeight);
            idList.add(nodeId);
        }

        return idList;
    }

    public static String calculateNodeId(List<Node> nodeList, int
totalWeight) {
        Assert.notEmpty(nodeList);
        Node maxCurrent = null;
        for (Node node : nodeList) {
            node.setCurrent(node.getCurrent() + node.getWeight());
            if (maxCurrent == null || node.getCurrent() >
maxCurrent.getCurrent()) {
                maxCurrent = node;
            }
        }
        // 前面验证了列表非空，纯是为了消除编辑器的警告
        assert maxCurrent != null;
        maxCurrent.setCurrent(maxCurrent.getCurrent() - totalWeight);
```

```
        return maxCurrent.getId();
    }

    /**
     * 求最大公约数
     */
    public static int gcd(int a, int b) {
        if (b == 0) {
            return a;
        }
        return gcd(b, a % b);
    }

    /**
     * 求数组的最大公约数
     */
    public static int gcdOfArray(List<Integer> arr) {
        Assert.notEmpty(arr);
        if (arr.size() == 1) {
            return arr.get(0);
        }
        return arr.stream().reduce(gcd(arr.get(0), arr.get(1)),
NodeCalculateUtil::gcd);
    }

}
```

想让Redis保持原子性执行多个命令，那就不得不使用lua脚本了

这个脚本的作用就是取出队列中下一个节点的id，并且记录一下下次请求的位置

```
local size = redis.call('LLEN', KEYS[1])
if size == 0 then
  error("empty list")
end
local i = redis.call('GET', KEYS[2])
if not i then
  i = 0
end
local v = redis.call('LINDEX', KEYS[1], i)
local newIndex = ( i + 1 ) % size
```

```
redis.call('SET', KEYS[2], newIndex)
return v

```

完事具备，剩下的我们只需要读取一下这个脚本，并对外提供 计算并保存节点方法 和 取下一个节点Id的方法

```
@EnableConfigurationProperties(SwrbProperties.class)
@RequiredArgsConstructor
@Slf4j
@Component
public class SmoothWeightedLoadBalancer implements InitializingBean {

    private final RedissonClient client;
    private final SwrbProperties properties;
    private final ResourceLoader resourceLoader;
    private String luaScript;
    private RList<String> rList;
    RScript script;

    public void saveNode(List<Node> nodeList) {
        checkList(nodeList);
        RLock lock = client.getLock(properties.getRedisPrefix() +
SAVE_NODE_LOCK_KEY);
        if (!lock.tryLock()) {
            throw new RuntimeException("正在保存节点队列");
        }
        try {
            rList.clear();
            RAtomicLong rAtomicLong =
client.getAtomicLong(properties.getRedisPrefix() + NODE_INDEX_KEY);
            rAtomicLong.set(0);
            List<String> idList = NodeCalculateUtil.calculateIdList(nodeList);
            rList.addAll(idList);
        } finally {
            lock.unlock();
        }
    }

    public String nextId() {
        if (rList.isEmpty()) {
            throw new RuntimeException("队列未初始化");
        }
        String indexKey = properties.getRedisPrefix() + NODE_INDEX_KEY;
```

```
        String nodeListKey = properties.getRedisPrefix() +
NODE_LIST_KEY;
        return script.eval(RScript.Mode.READ_WRITE, luaScript,
RScript.ReturnType.INTEGER,
                ListUtil.of(nodeListKey, indexKey));
    }


    @Override
    public void afterPropertiesSet() throws IOException {
        rList = client.getList(properties.getRedisPrefix() + NODE_LIST_KEY,
StringCodec.INSTANCE);
        script = client.getScript(StringCodec.INSTANCE);
        Resource resource =
resourceLoader.getResource("classpath:lua/nextId.lua");
        try (Reader reader = new
InputStreamReader(resource.getInputStream(), StandardCharsets.UTF_8))
{
            luaScript = FileCopyUtils.copyToString(reader);
        }
    }

    private void checkList(List<Node> nodeList) {
        Assert.notEmpty(nodeList);
        int maxSize = properties.getNodeListMaxSize();
        Assert.isTrue(nodeList.size() <= maxSize,
                String.format("节点队列最大长度: %s, 当前队列长度: %s",
maxSize, nodeList.size()));
        int maxWeight = properties.getNodeMaxWeight();

        nodeList.forEach(node -> {
            Assert.notBlank(node.getId());
            Assert.notNull(node.getWeight());
            Assert.notNull(node.getCurrent());
            Assert.checkBetween(node.getWeight(), 0, maxWeight,
                    String.format("节点最大权重: %s, 异常节点: %s",
maxWeight, node));
        });


    }

}
```

然后写个测试类看一下效果

```
@Slf4j
@SpringBootTest
class ApplicationTests {
    @Autowired
    private SmoothWeightedLoadBalancer loadBalancer;

    @Test
    void testNode() {
        loadBalancer.saveNode(list);
        for (int i = 0; i < 10; i++) {
            log.info("nextId:{}", loadBalancer.nextId());
        }

    }

    List<Node> list = ListUtil.of(
            Node.create("a", 2),
            Node.create("b", 3),
            Node.create("c", 1));
}
```

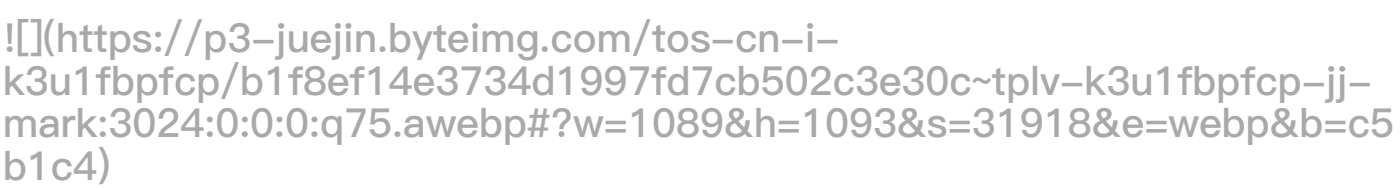
结果忘记截图了，不过执行结果符合预期 完美
结果忘记截图了，不过执行结果符合预期 完美

完整代码 [gitee.com/btkls/smoot...](http://cxyroad.com/"https://gitee.com/btkls/smooth-weighted-round-robin")

结尾
--

老马凭着这一手平滑加权轮询算法风靡全国，一时间所有的超市纷纷向他投来橄榄枝，他的程序让全国的老头都领上了自己的鸡蛋，自己也成功升职加薪出任CEO迎娶白富美走向人生巅峰。正在他春风得意的在海边度假时，一颗鸡蛋向他飞来，他转身一看原来他已经被愤怒的母鸡包围。由于他的程序让全国的

超市疯狂发鸡蛋，鸡蛋供不应求后养鸡场的管理员疯狂压榨母鸡日夜生蛋，终于引发了鸡因觉醒。愤怒的母鸡们最终赶走了压榨它们的人类，成为了地球的主宰~完



原文链接: https://juejin.cn/post/7366799820733890623