链滴

负载均衡的多种算法总结

作者：valarchie

原文链接：https://ld246.com/article/1568215395479

来源网站：链滴

负载均衡的多种算法总结

随机算法

先将服务器放进数组或者列表当中，通过JDK的随机算法，获取一个在数组有效范围内的下标，根据个随机下标访问对应服务器。由概率统计理论可以得知，随着客户端调用服务器的次数增多，其实际果越来越接近于平均分配请求到服务器列表中的每一台服务器。

代码：

```
 public String random(){
        String[] servers = {"server1", "server2", "server3"};
        // 将系统的当前时间作为种子获取一个随机器
        Random generator = new Random(System.currentTimeMillis());
        // 将服务器列表大小作为上界传入随机生成器
        int index = generator.nextInt(servers.length);
        return servers[index];
    }
```

加权随机算法

如果服务器的处理性能有高低的话，这时候就需要加权随机算法。加权随机算法也很简单，主要有两形式：

一种很简单的形式是按照服务器的权重，增大服务器列表中的个数。比如服务器A的权重是7，服务器的权重是3，那么服务器列表中就添加7个A服务器，添加3个B服务器。这时候进行随机算法的话，就有加权的效果了。

代码：

```
public String weightRandomA(){
        // 服务器列表
        String[] servers = {"serverA", "serverB"};
        // 权重
        int[] weights = {7, 3};

        List<String> weightServers = new ArrayList<>();
        // 根据权重大小往新的权重服务器列表里重复添加对应的服务器
        for (int i = 0; i < servers.length; i++) {
            for (int j = 0; j < weights[i]; j++) {
                weightServers.add(servers[i]);
            }
        }
        // 将系统的当前时间作为种子获取一个随机器
        Random generator = new Random(System.currentTimeMillis());
        // 将服务器列表大小作为上界传入随机生成器
        int index = generator.nextInt(weightServers.size());

        return weightServers.get(index);
    }
```

但是，这边也会出现一个问题，就是如果权重值很大的时候，权重服务器列表就会过大。另一种形式

将所有权重值进行相加，然后根据这个总权重值为随机数上界，进行随机抽取服务器。比如A服务器权重是2，B服务器的权重是3，C服务器的权重是5。总的权重值是10。在10当中取随机数。如果随数0到2之间的话，选择A服务器，随机数在3到5之间的话，选择B服务器，随机数在5到10之间的话选择C服务器。

代码：

```
public String weightRandomB(){
    // 服务器列表
    String[] servers = {"serverA", "serverB","serverC"};
    // 权重
    int[] weights = {2, 3, 5};
    // 总权重
    int totalWeight = 0;

    // 计算总权重
    for (int weight : weights) {
        totalWeight += weight;
    }
    // 将系统的当前时间作为种子获取一个随机器
    Random generator = new Random(System.currentTimeMillis());
    // 将总权重作为上界传入随机生成器  获取一个临时随机权重值
    int randomWeight = generator.nextInt(totalWeight);
    // 服务器列表下标
    int index = 0;
    // 递减  随机权重值  如果小于0的话，代表落入对应区间。根据得到的下标寻找服务器。
    for (int i = 0; i < weights.length; i++) {
        randomWeight -= weights[i];
        if (randomWeight <= 0) {
            index = i;
            break;
        }
    }

    return servers[index];
}
```

轮询算法

随机算法简单可行，但不够均衡，在极端情况下会造成一台服务器一直收到请求，另一个服务器一直收到请求。所以这时候就需要轮询算法。通过依次按顺序调用服务器列表中的服务器即可。例如服务列表中有ABC三台服务器，一个自增数字，每次自增完取3的余数，0的话取服务器A，1的话取服务器，2的话取服务器C即可。

代码：

```
public String roundRobin(){
    String[] servers = {"serverA", "serverB", "serverC"};
    // 用自增序列值 除 服务器列表的数量
    int currentIndex = serialNumber % servers.length;
    // 计算出此次服务器列表下标时，对自增序列+1
    //（当前使用类变量，实际开发可用Atomic原子变量）
    serialNumber++;
    return servers[currentIndex];
```

原文链接：负载均衡的多种算法总结

```
    }
```

普通加权轮询算法

如果考虑到不同服务器性能的话，就需要进行加权的轮询算法。

例如A服务器的权重为5，B服务器的权重为3，C服务器的权重为2。依次添加到服务器列表中，此时务器列表为[A,A,A,A,A,B,B,B,C,C]。依次轮询列表中的服务器即可实现加权轮询算法。

代码：

```
 public String weightRoundRobinA() {

        // 服务器列表
        String[] servers = {"serverA", "serverB","serverC"};
        // 权重
        int[] weights = {5, 3, 2};

        List<String> weightServers = new ArrayList<>();
        // 根据权重大小往新的权重服务器列表里重复添加对应的服务器
        for (int i = 0; i < servers.length; i++) {

            for (int j = 0; j < weights[i]; j++) {
                weightServers.add(servers[i]);
            }

        }

        // 用自增序列值 除 带权重服务器列表的数量
        int currentIndex = serialNumber % weightServers.size();
        // 计算出此次服务器列表下标时，对自增序列+1
        //（当前使用类变量，实际开发可用Atomic原子变量）
        serialNumber++;

        return weightServers.get(currentIndex);

    }
```

这种算法在权重值很大的时候列表会很长，此时可以取所有权重值的最大公约数，进行累加，落在对的区间时则取对应的服务器即可。例如服务器A的权重是10，服务器B的权重是3，服务器C的权重是2取公约数2，使用刚才的算法，每次自增序列递增公约数2即可。

平滑加权轮询算法

上面的加权轮询算法会导致连续的调用同一台服务器，此时请求分发显得很不均衡，总是需要按权重连续调用完同一台服务器之后才会调用接下来的服务器。

这时候就需要平滑加权算法。

假设服务器A配置权重为7，服务器B的配置权重为2，服务器配置权重为1。

总的权重值为10。平滑加权轮询的调度如下。每个服务器的有效权重为当前权重，与配置权重不同，效权重是根据上一轮再计算出来的结果。每一轮选取权重最大的服务器进行请求。被选取的节点，当有效权重减去总的权重值。下一轮开始前所有服务器的有效权重加上自己的配置权重。

- 服务器A权重：7
- 服务器B权重：2
- 服务器C权重：1
- 总权重：10
- 每轮各个服务器都会加上各自的配置权重

轮次	服务器A	服务器B	服务器C	前选中服务器（当前权重最大者）
第一轮	7	2	1	A(当权重-总权重=-3)
第二轮	4	2	2	A(当权重-总权重=-6)
第三轮	1	6	3	B(当权重-总权重=-4)
第四轮	8	-2	4	A(当权重-总权重=-2)
第五轮	5	0	5	A(当权重-总权重=-5)
第六轮	2	2	6	C(当权重-总权重=-4)
第七轮	9	4	-3	A(当权重-总权重=-1)
第八轮	6	6	-2	A(当权重-总权重=-4)
第九轮	3	8	-1	B(当权重-总权重=-2)
第十轮	10	0	1	A(当权重-总权重=0)

这样就不会出现连续重复的调用同一个服务器了。

代码：

```
public String smoothWeightRouncRobin() {

    // 服务器列表
    String[] servers = {"serverA", "serverB","serverC"};
    // 权重
    int[] weights = {7, 2, 1};
    // 总权重
    int totalWeight = 0;
    // 计算总权重
    for (int weight : weights) {
        totalWeight += weight;
    }

    int maxWeightIndex = 0;
```

```
        // currentWeights是一个类变量，保存三个服务器的当前权重
        // 寻找当前权重值最大的下标
        for (int i = 0; i < currentWeights.length; i++) {
            if (currentWeights[i] > currentWeights[maxWeightIndex]) {
                maxWeightIndex = i;
            }
        }

        // 当前权重最大者  要减去 总权重
        currentWeights[maxWeightIndex] = currentWeights[maxWeightIndex] - totalWeight;

        // 依次给每个服务器加上它的配置权重
        for (int i = 0; i < currentWeights.length; i++) {
            currentWeights[i] = currentWeights[i] + weights[i];
        }

        return servers[maxWeightIndex];

    }
```

哈希算法

可用ip地址或者请求的url进行哈希，请求分发到对应的服务器。

最小连接数算法

最小连接数法是根据服务器当前的连接情况进行负载均衡的，当请求到来时，会选取当前连接数最少一台服务器来处理请求。