负载均衡

负载均衡算法

1. 轮询：按照循环的顺序将请求分配给每个服务器，适用于服务器性能相近且负载均匀的情况。
2. 随机：随机选择一个服务器处理请求，适用于服务器性能相近且负载均匀的情况。
3. 加权轮询：权重越高的服务器，被分配处理的请求越多，适用于服务器性能不均的情况。
4. 加权随机：根据权重，随机分配服务器，适用于服务器性能不均的情况。
5. 最小连接数：根据连接数选择，选择连接数最小的服务器请求。适用于长连接。
6. IP hash：根据ip的hash值，请求会根据ip固定被分配到同一台服务器，适用于需要保持会话一致性的场景。

开发实现

负载均衡接口

```java
package com.yupi.yurpc.loadbalancer;

import com.yupi.yurpc.model.ServiceMetaInfo;

import java.util.List;
import java.util.Map;

/**
 * 负载均衡器（消费端使用）
 *
 * @author <a href="https://github.com/liyupi">程序员鱼皮</a>
 * @learn <a href="https://codefather.cn">鱼皮的编程宝典</a>
 * @from <a href="https://yupi.icu">编程导航学习圈</a>
 */
public interface LoadBalancer {

    /**
     * 选择服务调用
     *
     * @param requestParams       请求参数
     * @param serviceMetaInfoList 可用服务列表
     * @return
     */
    ServiceMetaInfo select(Map<String, Object> requestParams, List<ServiceMetaInfo> serviceMetaInfoList);
}

```

轮询

```java
package com.yupi.yurpc.loadbalancer;

import com.yupi.yurpc.model.ServiceMetaInfo;

import java.util.List;
import java.util.Map;
import java.util.concurrent.atomic.AtomicInteger;

/**
 * 轮询负载均衡器
 *
 * @author <a href="https://github.com/liyupi">程序员鱼皮</a>
 * @learn <a href="https://codefather.cn">鱼皮的编程宝典</a>
 * @from <a href="https://yupi.icu">编程导航学习圈</a>
 */
public class RoundRobinLoadBalancer implements LoadBalancer {

    /**
     * 当前轮询的下标
     */
    private final AtomicInteger currentIndex = new AtomicInteger(0);

    @Override
    public ServiceMetaInfo select(Map<String, Object> requestParams, List<ServiceMetaInfo> serviceMetaInfoList) {
        if (serviceMetaInfoList.isEmpty()) {
            return null;
        }
        // 只有一个服务，无需轮询
        int size = serviceMetaInfoList.size();
        if (size == 1) {
            return serviceMetaInfoList.get(0);
        }
        // 取模算法轮询
        int index = currentIndex.getAndIncrement() % size;
        return serviceMetaInfoList.get(index);
    }
}

```

随机

```java
package com.yupi.yurpc.loadbalancer;

import com.yupi.yurpc.model.ServiceMetaInfo;

import java.util.List;
import java.util.Map;
import java.util.Random;

/**
 * 随机负载均衡器
 *
 * @author <a href="https://github.com/liyupi">程序员鱼皮</a>
 * @learn <a href="https://codefather.cn">鱼皮的编程宝典</a>
 * @from <a href="https://yupi.icu">编程导航学习圈</a>
 */
public class RandomLoadBalancer implements LoadBalancer {

    private final Random random = new Random();

    @Override
    public ServiceMetaInfo select(Map<String, Object> requestParams, List<ServiceMetaInfo> serviceMetaInfoList) {
        int size = serviceMetaInfoList.size();
        if (size == 0) {
            return null;
        }
        // 只有 1 个服务，不用随机
        if (size == 1) {
            return serviceMetaInfoList.get(0);
        }
        return serviceMetaInfoList.get(random.nextInt(size));
    }
}

```

一致性hash

```java
package com.yupi.yurpc.loadbalancer;

import com.yupi.yurpc.model.ServiceMetaInfo;

import java.util.List;
import java.util.Map;
import java.util.TreeMap;

/**
 * 一致性哈希负载均衡器
 *
 * @author <a href="https://github.com/liyupi">程序员鱼皮</a>
 * @learn <a href="https://codefather.cn">鱼皮的编程宝典</a>
 * @from <a href="https://yupi.icu">编程导航学习圈</a>
 */
public class ConsistentHashLoadBalancer implements LoadBalancer {

    /**
     * 一致性 Hash 环，存放虚拟节点
     */
    private final TreeMap<Integer, ServiceMetaInfo> virtualNodes = new TreeMap<>();

    /**
     * 虚拟节点数
     */
    private static final int VIRTUAL_NODE_NUM = 100;

    @Override
    public ServiceMetaInfo select(Map<String, Object> requestParams, List<ServiceMetaInfo> serviceMetaInfoList) {
        if (serviceMetaInfoList.isEmpty()) {
            return null;
        }

        // 构建虚拟节点环
        for (ServiceMetaInfo serviceMetaInfo : serviceMetaInfoList) {
            for (int i = 0; i < VIRTUAL_NODE_NUM; i++) {
                int hash = getHash(serviceMetaInfo.getServiceAddress() + "#" + i);
                virtualNodes.put(hash, serviceMetaInfo);
            }
        }

        // 获取调用请求的 hash 值
        int hash = getHash(requestParams);

        // 选择最接近且大于等于调用请求 hash 值的虚拟节点
        Map.Entry<Integer, ServiceMetaInfo> entry = virtualNodes.ceilingEntry(hash);
        if (entry == null) {
            // 如果没有大于等于调用请求 hash 值的虚拟节点，则返回环首部的节点
            entry = virtualNodes.firstEntry();
        }
        return entry.getValue();
    }


    /**
     * Hash 算法，可自行实现
     *
     * @param key
     * @return
     */
    private int getHash(Object key) {
        return key.hashCode();
    }
}
```

可配置可扩展实现

1. 常量
2. 工厂模式
3. META—INFO下新建配置文件
4. 全局配置类添加负载均衡配置

Java

```java
package com.yupi.yurpc.loadbalancer;

/**
 * 负载均衡器键名常量
 *
 * @author <a href="https://github.com/liyupi">程序员鱼皮</a>
 * @learn <a href="https://codefather.cn">鱼皮的编程宝典</a>
 * @from <a href="https://yupi.icu">编程导航学习圈</a>
 */
public interface LoadBalancerKeys {

    /**
     * 轮询
     */
    String ROUND_ROBIN = "roundRobin";

    String RANDOM = "random";

    String CONSISTENT_HASH = "consistentHash";

}

```

Java

```java
package com.yupi.yurpc.loadbalancer;

import com.yupi.yurpc.spi.SpiLoader;

/**
 * 负载均衡器工厂（工厂模式，用于获取负载均衡器对象）
 *
 * @author <a href="https://github.com/liyupi">程序员鱼皮</a>
 * @learn <a href="https://codefather.cn">编程宝典</a>
 * @from <a href="https://yupi.icu">编程导航知识星球</a>
 */
public class LoadBalancerFactory {

    static {
        SpiLoader.load(LoadBalancer.class);
    }

    /**
     * 默认负载均衡器
     */
    private static final LoadBalancer DEFAULT_LOAD_BALANCER = new RoundRobinLoadBalancer();

    /**
     * 获取实例
     *
     * @param key
     * @return
     */
    public static LoadBalancer getInstance(String key) {
        return SpiLoader.getInstance(LoadBalancer.class, key);
    }

}

```

Java

```
roundRobin=com.yupi.yurpc.loadbalancer.RoundRobinLoadBalancer
random=com.yupi.yurpc.loadbalancer.RandomLoadBalancer
consistentHash=com.yupi.yurpc.loadbalancer.ConsistentHashLoadBalancer

```

```java
/**
 * 服务代理（JDK 动态代理）
 *
 * @author <a href="https://github.com/liyupi">程序员鱼皮</a>
 * @learn <a href="https://codefather.cn">编程宝典</a>
 * @from <a href="https://yupi.icu">编程导航知识星球</a>
 */
public class ServiceProxy implements InvocationHandler {

    /**
     * 调用代理
     *
     * @return
     * @throws Throwable
     */
    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        // 指定序列化器
        final Serializer serializer = SerializerFactory.getInstance(RpcApplication.getRpcConfig().getSerializer());

        // 构造请求
        String serviceName = method.getDeclaringClass().getName();
        RpcRequest rpcRequest = RpcRequest.builder()
                .serviceName(serviceName)
                .methodName(method.getName())
                .parameterTypes(method.getParameterTypes())
                .args(args)
                .build();
        try {
            // 从注册中心获取服务提供者请求地址
            RpcConfig rpcConfig = RpcApplication.getRpcConfig();
            Registry registry = RegistryFactory.getInstance(rpcConfig.getRegistryConfig().getRegistry());
            ServiceMetaInfo serviceMetaInfo = new ServiceMetaInfo();
            serviceMetaInfo.setServiceName(serviceName);
            serviceMetaInfo.setServiceVersion(RpcConstant.DEFAULT_SERVICE_VERSION);
            List<ServiceMetaInfo> serviceMetaInfoList = registry.serviceDiscovery(serviceMetaInfo.getServiceKey());
            if (CollUtil.isEmpty(serviceMetaInfoList)) {
                throw new RuntimeException("暂无服务地址");
            }

            // 负载均衡
            LoadBalancer loadBalancer = LoadBalancerFactory.getInstance(rpcConfig.getLoadBalancer());
            // 将调用方法名（请求路径）作为负载均衡参数
            Map<String, Object> requestParams = new HashMap<>();
            requestParams.put("methodName", rpcRequest.getMethodName());
            ServiceMetaInfo selectedServiceMetaInfo = loadBalancer.select(requestParams, serviceMetaInfoList);

            // rpc 请求
            RpcResponse rpcResponse = VertxTcpClient.doRequest(rpcRequest, selectedServiceMetaInfo);
            return rpcResponse.getData();
        } catch (Exception e) {
            throw new RuntimeException("调用失败");
        }
    }
}
```