## 分布式计算选做3实验报告

**一、实验题目**

1. 实现⼀个基于⼀致性哈希算法的分布式键值(K-V)数据存储系统

2. 系统提供⼀个客⼾端SDK,该SDK⽀持两类操作：

a) put(key,value):写⼀个键值对，key是定⻓字符串；value是⻓度受限的变⻓字符

串

b) get(key):返回指定key对应的value.如果不存在则返回null

3. 在客⼾端SDK中：以key为输⼊，利⽤⼀致性哈希算法计算出负责该key的节点标

识，然后将put/get操作发送给该节点

4. 不同节点可以在单机上⽤不同的进程(在不同的端⼝号监听)或虚拟机模拟

5. 将服务器节点标识列表作为客⼾端SDK的初始化参数

6. 键值对数据保存在内存即可

7. ⽀持增删节点

二、**实验原理**

一致性哈希的原理可以概括为以下几个步骤：

1. 建立哈希环：将哈希空间表示为一个环状结构，通常使用一个 32 位或 64 位的哈希空间，例如使用 0 到 2^32-1 之间的整数表示。
2. 节点映射：将每个节点（服务器）通过哈希函数映射到哈希环上的一个位置。常见的哈希函数有 MD5、SHA-1、SHA-256 等，将节点的唯一标识（如 IP 地址或名称）映射为一个哈希值。
3. 数据映射：同样地，将要存储的数据通过哈希函数映射到哈希环上的一个位置。数据的唯一标识（如键）被映射为一个哈希值。
4. 寻找节点：当需要存储或获取数据时，根据数据的哈希值在哈希环上顺时针寻找离该哈希值最近的节点。即从数据的哈希值位置出发，按顺时针方向寻找下一个节点，直到找到一个节点为止。
5. 数据复制：为了提高数据的可靠性和冗余度，可以将数据复制到相邻的多个节点上。通常，从数据的哈希值位置开始，顺时针寻找下一个节点作为数据的主节点，并将数据复制到主节点的相邻节点上。

6.通过这种方式，一致性哈希实现了数据的均匀分布和节点的动态扩缩容。当节点加入或离开系统时，只会影响到环上相邻的节点和它们负责的数据，而不会对整个系统造成较大的影响。这样可以降低节点迁移的开销，同时保持数据的高可用性和负载均衡。

**三、设计思路**

1. 由于为了模拟多节点，我们采用多线程来进行程序设计。
2. 在选择哈希算法时选择MD5算法， 这种算法可以唯一地标识输入数据。
3. 根据题目要求，我设计了五个类来解决问题：

Node类：定义了Node的数据类型，定义节点的属性，并提供了可供调用的方法来查询节点的属性。

ConsistentHash类：初始化时将已创建的节点加入到哈希环中，另外定义了以下函数：

addNode():用来添加节点到哈希环中。

removeNode():将哈希环中指定ip的节点删除。

getHash():将输入的key值通过MD5算法转换为哈希值。

getNode():用于将输入的key值映射到对应的哈希节点。

Server类：定义节点的服务端，设计了可以将键值对插入的服务功能，用户不需要知道节点具体拓扑结构，只需将键值对上传，由服务器来执行一致性哈希的插入操作。

Client类：赋予了用户可以增删节点的权力，暴露了server类中的put()方法接口，用户可以调用server对象的插入方法，将数据按照一致性哈希来插入数据。

Main类：主要用于执行和调用方法来测试功能。

1. **代码实现**

Node类

public class Node {

    private final String ipAddress;

    private final int port;

    public Node(String ipAddress, int port) {

        this.ipAddress = ipAddress;

        this.port = port;

    }

    public String getIpAddress() {

        return ipAddress;

    }

    public int getPort() {

        return port;

    }

}

ConsistentHash类：

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.util.List;

import java.util.SortedMap;

import java.util.TreeMap;

public class ConsistentHashing {

    private final SortedMap<Integer, Node> ring;

    private final int virtualNodes;

    public ConsistentHashing(List<Node> nodes, int virtualNodes) {

        this.virtualNodes = virtualNodes;

        this.ring = new TreeMap<>();

        for (Node node : nodes) {

            addNode(node);

            System.out.println("节点 " + node.getIpAddress() + ":" + node.getPort() + " 添加成功！\n");

            System.out.println("当前哈希环状态为： " + ring + "\n");

        }

    }

    public void addNode(Node node) {

        for (int i = 0; i < virtualNodes; i++) {

            String virtualNodeId = node.getIpAddress() + ":" + node.getPort() + "-" + i;

            int hash = getHash(virtualNodeId);

            System.out.println("虚拟节点： " + virtualNodeId + " 为 " + hash);

            ring.put(hash, node);

        }

    }

    public void removeNode(Node node) {

        for (int i = 0; i < virtualNodes; i++) {

            String virtualNodeId = node.getIpAddress() + ":" + node.getPort() + "-" + i;

            int hash = getHash(virtualNodeId);

            ring.remove(hash);

            System.out.println("虚拟节点 " + virtualNodeId + "移除成功！");

        }

    }

    private int getHash(String key) {

        try {

            MessageDigest messageDigest =

                    MessageDigest.getInstance("MD5");

            messageDigest.update(key.getBytes());

            byte[] digest = messageDigest.digest();

            int hash = 0;

            for (int i = 0; i < 4; i++) {

                hash <<= 8;

                hash |= ((int) digest[i]) & 0xFF;

            }

            return hash & 0x7FFFFFFF;

        } catch (NoSuchAlgorithmException e) {

            e.printStackTrace();

        }

        return 0;

    }

    public Node getNode(String key) {

        if (ring.isEmpty()) {

            return null;

        }

        int hash = getHash(key);

        SortedMap<Integer, Node> tailMap = ring.tailMap(hash);

        if (tailMap.isEmpty()) {

            System.out.println(hash + " 被映射于：" + ring.get(ring.firstKey()));

            return ring.get(ring.firstKey());

        } else {

            System.out.println(hash + "被映射于：" + tailMap.get(tailMap.firstKey()));

            return tailMap.get(tailMap.firstKey());

        }

    }

    public SortedMap<Integer, Node> getRing() {

        return ring;

    }

    public int getVirtualNodes() {

        return virtualNodes;

    }

}

Server类：

import java.util.HashMap;

import java.util.Map;

public class Server {

    private final String ipAddress;

    private final int port;

    private final Map<String, String> keyValueStore;

    public Server(String ipAddress, int port) {

        this.ipAddress = ipAddress;

        this.port = port;

        this.keyValueStore = new HashMap<>();

    }

    public void put(String key, String value) {

        keyValueStore.put(key, value);

        System.out.println("已成功将 " + key + " - " + value + "存储于" + ipAddress + ":" + port);

    }

    public String get(String key) {

        String value = keyValueStore.get(key);

        System.out.println("成功从 "+ipAddress + ":" + port + ": " + "读取键值对 " + key + " - " + value);

        return value;

    }

    public String getIpAddress() {

        return ipAddress;

    }

    public int getPort() {

        return port;

    }

}

Client类：

import java.util.ArrayList;

import java.util.List;

import java.util.TreeMap;

public class Client {

    private final ConsistentHashing hashStore;

    private final List<Server> servers;

    public Client(List<Node> nodes, int virtualNodes) {

        this.servers = new ArrayList<>();

        for (Node node : nodes) {

            servers.add(new Server(node.getIpAddress(),

            node.getPort()));

        }

        this.hashStore = new ConsistentHashing(nodes, virtualNodes);

        System.out.println("当前哈希环为 " + hashStore + "\n");

    }

    public void put(String key, String value) {

        Node node = hashStore.getNode(key);

        Server server = findServer(node);

        server.put(key, value);

    }

    public String get(String key) {

        Node node = hashStore.getNode(key);

        Server server = findServer(node);

        return server.get(key);

    }

    private Server findServer(Node node) {

        for (Server server : servers) {

            if (server.getIpAddress().equals(node.getIpAddress()) && server.getPort() == node.getPort()) {

                return server;

            }

        }

        return null;

    }

    public void addNode(Node node) {

        servers.add(new Server(node.getIpAddress(), node.getPort()));

        hashStore.addNode(node);

        System.out.println("节点 " + node.getIpAddress() + ":" + node.getPort() + " 添加成功！ .\n");

        System.out.println("当前哈希环 " + hashStore.getRing() +"\n");

    }

    public void removeNode(Node node) {

        Server serverToRemove = null;

        for (Server server : servers) {

            if (server.getIpAddress().equals(node.getIpAddress()) &&

                    server.getPort() == node.getPort()) {

                serverToRemove = server;

                break;

            }

        }

        if (serverToRemove != null) {

            servers.remove(serverToRemove);

        }

        hashStore.removeNode(node);

    }

    public TreeMap<Integer, Node> getRing() {

        return (TreeMap<Integer, Node>) hashStore.getRing();

    }

}

Main类：

import java.util.ArrayList;

import java.util.List;

public class Main {

    public static void main(String[] args) {

        List<Node> nodes = new ArrayList<>();

        Node node1 = new Node("Node1", 8080);

        Node node2 = new Node("Node2", 8081);

        Node node3 = new Node("Node3", 8082);

        nodes.add(node1);

        nodes.add(node2);

        nodes.add(node3);

*// 虚拟节点为3*

        String key1 = "Teacher1";

        String key2 = "Teacher2";

        String key3 = "Student1";

        String key4 = "Student2";

        String key5 = "Student3";

        Client client = new Client(nodes, 3);

        client.put(key1, "19711110");

        client.put(key2, "19710205");

        client.put(key3, "20020627");

        String value1 = client.get(key1);

        String value2 = client.get(key2);

        String value3 = client.get(key3);

        System.out.println("value1: " + value1);

        System.out.println("value2: " + value2);

        System.out.println("value3: " + value3);

        Node node4 = new Node("Node4", 8083);

        client.addNode(node4);

        client.put(key4, "20020815");

        client.put(key5, "20020329");

        String value4 = client.get(key4);

        String value5 = client.get(key5);

        System.out.println("value4: " + value4);

        System.out.println("value5: " + value5);

        client.removeNode(node2);

        client.removeNode(node3);

        client.removeNode(node4);

        System.out.println("当前哈希环： " + client.getRing() + "\n");

        String deletedV2 = client.get(key4);

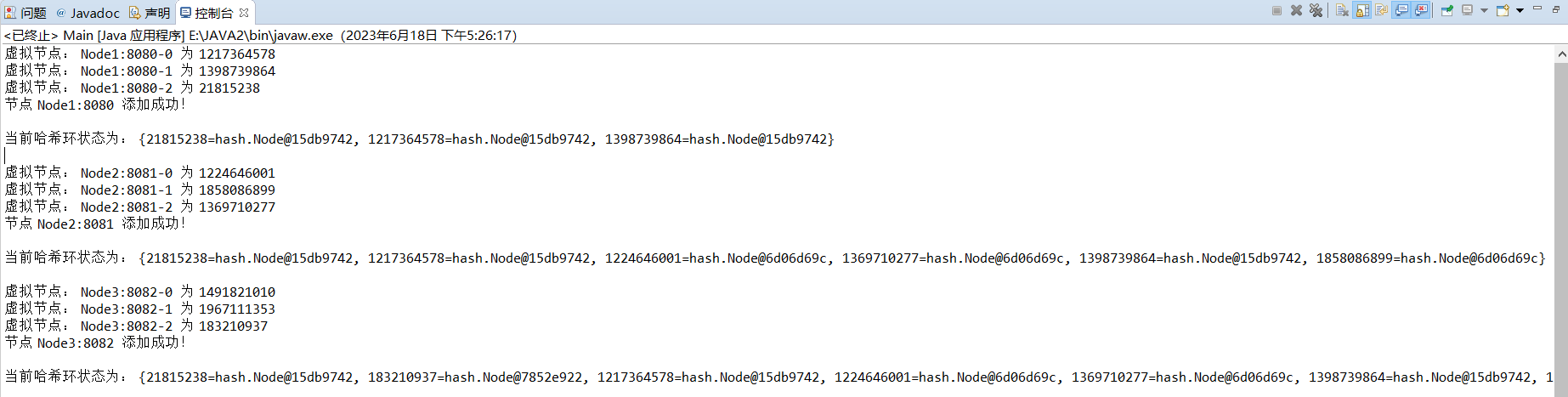
        System.out.println("删除节点：" + deletedV2);

    }

}

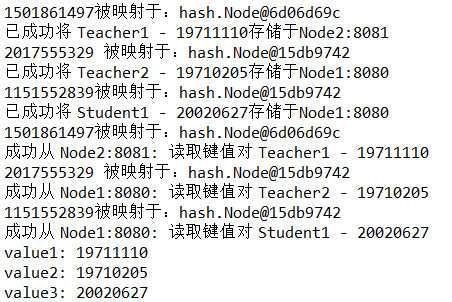
**五、实验结果**

1.首先创建三个哈希节点：



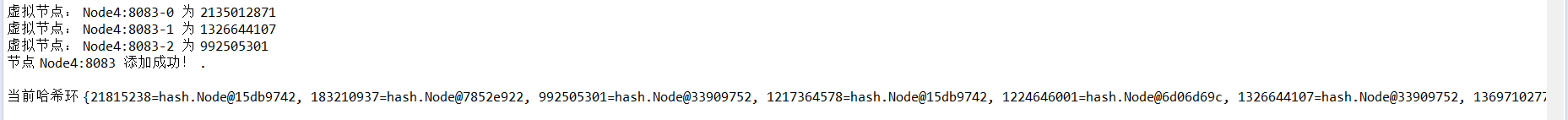
可以看到三个节点被先后创建，并分别输出了各自的详细信息。

2.键值对的存储与读取：



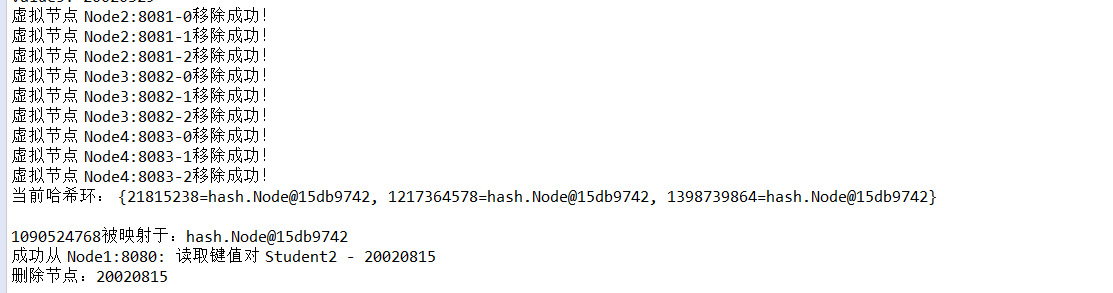
可以看到将老师与学生的生日信息分别存储与读取成功。

3.创建了第四个节点：



4.向哈希环中添加了两组键值对，然后将它们读取并打印。 

5.移除三个虚拟节点后，有部分数据重新映射了：



**六、实验心得与感悟**

通过本次实验，我认识到一致性哈希算法在分布式系统中的负载均衡和扩展性方面具有很大的优势。通过将数据和节点映射到一个哈希环上，可以实现数据的均匀分布和节点的动态扩缩容。这样可以避免传统哈希算法中的数据热点问题，使得数据在节点间更加均衡地分布，提高系统的整体性能和吞吐量。

通过本次实验，我对于分布式计算的认识更加深入，对于分布式计算的应用有了新的认识。