Java实验报告

软件学院 软件工程4班 2113850 李鹏

——实验题目：Java实现LRU缓存模型

1. 本次实验实现了一个基于LRU算法的缓存，用于保存键值对。

具体实现思路如下：

1. 定义一个泛型类LRUCache<K, V>，其中使用Java内置的哈希表Map<K, V>来保存数据。

2. 通过传递一个整数参数capacity作为缓存容量，在构造函数中初始化一个LinkedHashMap<K, V>来保存缓存数据，并且添加了一个removeEldestEntry方法，用于实现LRU算法的缓存淘汰。这里的removeEldestEntry方法会在LinkedHashMap中添加新元素后被调用，通过返回true或false来决定是否删除最老的元素。

3. 在主函数中，通过Scanner对象读入缓存容量capacity和操作次数n。然后定义一个二维数组str来保存每次操作的参数，同时创建一个LRUCache对象lruCache作为缓存。

4. 循环读入每次操作的参数，并根据操作类型分别调用LRUCache类中相应的方法来实现缓存功能。对于put操作，先将多个键值对按照分号分隔成一个字符串数组keyValuePairs，再将每个键值对按照逗号分隔成一个键值数组keyValue，并判断键值对是否合法。若合法则将其添加到LRUCache中。对于get操作，先将多个键按照分号分隔成一个字符串数组keys，然后循环遍历每个键，并获取该键对应的值，并将结果字符串output根据获取到的值进行拼接，最后输出结果字符串。

5. 主函数执行完毕，关闭Scanner对象。

1. 总的来说，这个程序的时间复杂度可以分为两部分：

1. LRUCache类中的访问方式put、get、delete方法的时间复杂度均为O(1)，因为使用了Java内置的哈希表Map，它的查找、插入、删除操作都是常数级别的时间复杂度。removeEldestEntry方法的时间复杂度也是O(1)。

2. 主函数中循环执行次数为n，因此主函数的时间复杂度为O(n)。

因此，该程序的总时间复杂度为O(n)，该算法的空间复杂度是O(capacity)，因为缓存容量为capacity。

1. 源代码：

import java.util.LinkedHashMap;

import java.util.Map;

import java.util.Scanner;

//定义一个泛型类LRUCache，其类型参数K,V分别表示缓存的键和值

class LRUCache<K, V> {

Map<K, V> map; //使用Java内置的哈希表Map来保存数据

//构造函数，接收一个整数参数capacity，表示缓存容量，用于初始化map

public LRUCache(final int capacity) {

this.map = new LinkedHashMap<K, V>(capacity, 0.75f, true) { //使用LinkedHashMap来保存缓存数据

//这里添加了一个removeEldestEntry方法，用于实现LRU算法的缓存淘汰

protected boolean removeEldestEntry(Map.Entry eldest) {

return size() > capacity; //如果当前容量超过了缓存容量，则删除最近最少使用的元素

}

};

}

//向缓存中添加一个键值对

public synchronized void put(K key, V value) {

map.put(key, value);

}

//从缓存中获取指定键的值

public synchronized V get(K key) {

if (map.containsKey(key))

return map.get(key);

else

return null; //如果缓存中不存在该键，则返回空

}

//从缓存中删除指定键的值

public synchronized void delete(K key) {

map.remove(key);

}

}

public class cache {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int capacity = sc.nextInt(); //读入缓存容量

int n = sc.nextInt(); //读入操作次数

String[][] str = new String[n][2]; //定义一个二维数组str，用于保存每次操作的参数

LRUCache<String, String> lruCache = new LRUCache<String, String>(capacity); //创建一个LRUCache对象，用于实现缓存

//循环读入每次操作的参数

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

for (int j = 0; j < 2; j++) {

str[i][j] = sc.next();

}

if (str[i][0].equals("put")) { //如果操作是向缓存中添加元素

String[] keyValuePairs = str[i][1].split(";"); //将多个键值对按照分号分隔

for (String keyValuePair : keyValuePairs) {

String[] keyValue = keyValuePair.split(","); //将每个键值对按照逗号分隔

if (keyValue.length == 2) { //判断键值对是否合法

lruCache.put(keyValue[0], keyValue[1]); //将键值对添加到LRUCache中

}

}

} else if (str[i][0].equals("get")) { //如果操作是从缓存中获取元素

String[] keys = str[i][1].split(";"); //将多个键按照分号分隔

String output = ""; //定义一个字符串变量output，用于保存结果字符串

for (String key : keys) { //循环遍历每个键

String value = lruCache.get(key); //获取该键对应的值

if (value == null) { //如果值为空，则表示该键不存在于缓存中

output += "null;"; //将"null;"添加到结果字符串中

} else {

output += (value + ";"); //将该键的值添加到结果字符串中

}

}

System.out.println(output); //输出结果字符串

}

}

sc.close(); //关闭Scanner对象

}

}