As one of the important members of data structure in Java, HashMap has high performance in both insertion and query. Recently, I have learnt to know its power when I was working on the project. HashMap provides both high flexibility and scalability for font and back end data interactions that cannot be replaced by other data structures. Its key-value pair structure naturally provides good support for Json transformation. This article mainly analysis the theory

Hash is an algorithm, which can map a binary value with random length to a binary value with fixed length. In a HashMap, hash algorithm is mainly used to calculate the index that its value needs to be stored in. In one word, the storage process of a value is based on the hash value of its key.

The performance of a hash algorithm depends on whether it can store data into an array with good distribution so that collisions can be avoided.

There are many hash functions existed, and I will take division method as an example. First, define the length of an array, 16 for example (16 is also the default length of HashMap in JDK). Therefore, the index should be key mod m, and the value of m should be the maximum prime number that is less than the length of the array. So m is 13 in this case. Such algorithm would lead to situation like this: we might get same index for different given keys. If that’s the case, we need to handle collisions.

Ps. The hash algorithm in JDK is much more complicated. Above is just an example for readers to understand more easily.

## 2.3 处理冲突

- 线性探测法

当冲突产生时，查找下一个索引是否被占用，如果没有，则把数据存到该索引上。

- 链表形式

由于在HashMap中，单个数据是以entry的形式存储的，而entry中包含了key，value和next指针。那么当冲突产生时，我们就把原先存放到这个位置的数据取出来，然后在这个位置存放新的数据，并且把新数据的next指针设为原数据，也就是说链表头位置的数据永远是最新的数据。

## 2.3 Handling collisions

There are many solutions to handle collisions, and in this article I will introduce two methods and I will demonstrate how to solve this issue with one of them.

* Linear Probing

When collision happens, find whether next index has been occupied or not, if not then store data into that index. If it is already occupied, repeat previous process until it finds a slot.

* Linked list

Since data is stored in the form of entry object in HashMap, and an entry contains key, value and next pointer. Therefore we can handle collisions using LinkedList form, that is taking out the original data in this slot and place new data into it and then set the next pointer to the original data, which means the head of the this linkedlist is always the latest data.

## 3.1 Map接口

HashMap的顶层接口是Map，那么我们自己实现的Map也需要一个接口，在这里我定义接口的名称为MyMap<K,V>。这个接口中应该含有的方法包括：

- put(K k,V v)

- get(K k)

- size()

和一个内部接口

- Entry<K,V>

这个内部接口中包含两个方法

- getKey()

- getValue()

代码如下

## 3.1 Map Interface

The top interface of HashMap is Map, which means we need our own interface if we want to implement our own map. The map interface of our HashMap is defined as MyMap<K,V> and it should have these three functions:

- put(K k,V v)

- get(K k)

- size()

and an internal interface

- Entry<K,V>

It should contains two functions

- getKey()

- getValue()

## 3.2 内部类Entry的实现

接口设计完毕之后，我们需要创建一个类来实现这个接口的方法。我创建了一个名为MyHashMap的类实现MyMap接口。

这里我们需要一个内部类来实现MyMap的内部接口，内部类的实例对象即数组中存储的entry对象，所以我们需要定义三个成员变量，分别是K，V和Next。next的类型就是entry本身，因为它指向的是下一个entry对象。

内部类代码如下

## 3.2 Implementation of internal class Entry

Once the interface is designed, we need to create a class to implement all its functions. The class MyHashMap is created to implement MyMap interface.

An internal class is also necessary for the implementation of internal interface of MyMap. The instance of this internal class is entry, which will be stored into the array. This class should have three global variables, which are K, V and Next pointer. The type of next is entry itself because it points to the next entry object.

Java code:

## 3.3 定义成员变量

HashMap中含有以下几个成员变量：

- 默认数组长度

- 默认负载因子

- Entry数组

- HashMap的大小

## 3.3 Define global variables

A hashmap contains the following global variables:

- default length of the array

- default loading factor

- an entry array

- the size of this hashmap

## 3.4 定义构造方法

在HashMap中默认数组长度和默认负载因子都是可以自定义的，那么我们定义一个可以自定义数组长度和负载因子的构造方法。

## 3.4 Define constructor

Since the default length and default loading factor can be customized in HashMap, we can define a constructor with array length and loading factor as parameters.

If parameters are not specified, we use default value

## 3.5定义哈希函数

上文已经提过了，哈希函数我们使用除留取余法。定义一个整型m，m的取值应该是一个比数组长度小的最大质数，为了简化算法我取数组的长度作为m的值。以key的哈希值模于m，得到index的值并且返回。

## 3.5 Define hash function

It is mentioned above that we are using mod method for our hash function. First define an int m, the value of m should be the maximum prime number that is less than the length of its array. I set the value of m to be the length of the array to simplify the algorithm.

Ternary Operator is used at the end to ensure that the value of this index is positive.

## 3.6 实现put方法

首先，我们需要通过哈希算法得到数组的下标，然后把一个包含键值对以及next指针的entry对象存到该位置中。

在存入数组之前我们需要判断当前索引中是否已经存在数据。根据不同情况，做出不同的存储处理，代码中的注释有详细的解释。

## 3.6 Implement put function

First, we need calculate the index of the array using hash algorithm, and then store the entry object that contains key, value and next pointer to that slot.

Before we do that we need to determine if that slot is occupied. It should be handled differently according to the situation. Detailed explanations are given in the comments.

```Java

public V put(K k, V v) {

//根据key和哈希算法算出数组下标

int index = getKey(k);

Entry<K, V> entry = table[index];

//判断entry是否为空

if (entry == null) {

/\*

\* 如果entry为空，则代表当前位置没有数据。

\* new一个entry对象，内部存放key，value。

\* 此时next指针没有值，因为这个位置上只有一个entry对象

\* \*/

table[index] = new Entry(k, v, null);

//map的大小加1

size++;

} else {

/\*

\* 如果entry不为空，则代表当前位置已经有数据了

\* new一个entry对象，内部存放key，value。

\* 把next指针设置为原本的entry对象并且把数组中的数据替换为新的entry对象

\* \*/

table[index] = new Entry<K, V>(k, v, entry);

}

return table[index].getValue();

}

```

## 3.7 实现HashMap的扩容

HashMap的扩容机制是：当map的大小大于默认长度\*默认负载因子，那么数组的长度会翻倍，数组中的数据会重新散列然后再存放。那么在原先的put方法中，需要先判断是否达到扩容的标准在进行执行下面的代码。如果达到扩容的标准则需要调用扩容的方法。代码如下：

## 3.7 Implement resize mechanism of HashMap

The principle of resize in hashmap is that when the size of map is greater than default length \* loading factor, the length of the array will be doubled and data in the array will be rehashed and restored. Therefore, in previous put function, we need to first determine if the size of hashmap has reach the boundary of resize, and then execute rest of the code.

```Java

//数组的扩容

private void expand() {

//创建一个大小是原来两倍的entry数组

Entry<K, V>[] newTable = new Entry[2 \* defaultLength];

//重新散列

rehash(newTable);

}

//重新散列的过程

private void rehash(Entry<K,V>[] newTable) {

//创建一个list用于装载HashMap中所有的entry对象

List<Entry<K, V>> list = new ArrayList<Entry<K, V>>();

//遍历整个数组

for(int i=0; i<table.length;i++) {

//如果数组中的某个位置没有数据，则跳过

if (table[i] == null) {

continue;

}

//通过递归的方式将所有的entry对象装载到list中

findEntryByNext(table[i],list);

if (list.size() > 0) {

//把size重置

size = 0;

//把默认长度设置为原来的两倍

defaultLength = 2 \* defaultLength;

table = newTable;

for (Entry<K, V> entry : list) {

if (entry.next != null) {

//把所有entry的next指针置空

entry.next = null;

}

//对新table进行散列

put(entry.getKey(), entry.getValue());

}

}

}

}

private void findEntryByNext(Entry<K, V> entry,List<Entry<K, V>> list ) {

if (entry != null && entry.next != null) {

list.add(entry);

//递归调用

findEntryByNext(entry.next,list);

}else {

list.add(entry);

}

}

```

## 3.8 实现get方法

上文提到了，由于hash算法可能会导致相同的索引中包含了不同的entry对象，我们需要通过对比key值的方式来找到我们真正要的那个entry对象。代码如下：