Tutorial for STLite-LinkedHashMap

(1) : HashMap

LinkedHashMap在本质上是一个实现了按照插入顺序访问元素的HashMap，所以在介绍LinkedHashMap之前，我们需要先了解HashMap。

HashMap (哈希表) 是一个使用关键词索引的数据结构，支持关键词对应元素的插入，和查询关键词对应的元素，并且这些操作的期望时间复杂度都是线性的。

1、哈希的相关概念

Hash 就是把任意长度的输入，通过某种哈希算法，变换某种与之对应的输出 (通常是整数)，该输出我们称为哈希值。不同的输入可能会散列成相同的输出，从而不可能从散列值来唯一的确定输入值，这种情况我们称之为哈希碰撞。(在本次作业中，推荐使用 `std::hash` 来实现哈希函数)

2、使用哈希算法构建支持快速查找、删除的数据结构

有了哈希函数，我们最自然的想法就是构建一个数组，数组的下标对应着哈希值。每当我们插入一对 $(Key, Value)$ 的时候，我们先通过哈希函数对 Key 进行处理得到哈希值 $hash$，然后将 $Value$ 储存到数组的第 $hash \ mod \ Listsize$ 位，但是我们会发现一个问题，当遇到之前所说的哈希碰撞问题的时候，我们没有办法在数组的同一个下标处储存两个元素，那么比较自然的想法对于每一个下标我们都开一个链表，如果遇到冲突问题，我们就在链表的末尾添加这个元素。我们查询的时候对于一个给定的 Key 首先先进行哈希得到哈希值 $hash$，再在第 $hash \ mod \ Listsize$ 个链表中依次寻找 Key 所对应的元素，这就实现了一个固定大小的哈希表。

3、一些优化

我们需要实现的是一个封装好的数据结构供其他人使用，我们在开发的时候实际上并不知道数据规模的大小，所以对于哈希表大小的选择是一个非常关键的问题，当我们选择的哈希表大小比较小的时候，链表长度可能会比较长，查询复杂度会退化，当哈希表大小比较大的时候，会占用很多无用空间。这也启发我们可以动态的改变哈希表的大小。首先引入两个参数$Capacity, LoadFactor$，分别是容量和负载因子，代表着哈希表的大小，和对于某个特定的容量，我们所能接受的最多元素个数占容量的比例。我们一开始可以选择一个比较小的容量，当元素个数大于$Capacity * LoadFactor$ 时，我们再增大我们的 $Capacity$ ，使得我们的数据结构能够保持良好效率的同时，不占用过多空间，具体参数大家可以根据自己实现的数据结构去进行调整。

(2) : LinkedHashMap

LinkedHashMap需要在实现HashMap功能的基础上，再进行插入顺序的维护，使得我们可以按照插入顺序来访问元素。这项功能的实现比较简单，只需要维护一个双向链表，每次添加的元素除了插入HashMap外，也需要插入到链表的末尾，这样我们按照链表顺序访问元素就可以实现这项功能。

P.S.

我们实现的LinkedHashMap在底层原理上和Java_LinkedHashMap基本上是一致的，如果对于算法有什么不了解的地方，可以参考Java_LinkedHashMap的实现。