LRU缓存实现(Java)

LRU是Least Recently Used 的缩写，翻译过来就是“最近最少使用”，LRU缓存就是使用这种原理实现，简单的说就是缓存一定量的数据，当超过设定的阈值时就把一些过期的数据删除掉，比如我们缓存10000条数据，当数据小于10000时可以随意添加，当超过10000时就需要把新的数据添加进来，同时要把过期数据删除，以确保我们最大缓存10000条，那怎么确定删除哪条过期数据呢，采用LRU算法实现的话就是将最老的数据删掉，废话不多说，下面来说下Java版的LRU缓存实现

Java里面实现LRU缓存通常有两种选择，一种是使用LinkedHashMap，一种是自己设计数据结构，使用链表+HashMap

LRU Cache的LinkedHashMap实现

LinkedHashMap自身已经实现了顺序存储，默认情况下是按照元素的添加顺序存储，也可以启用按照访问顺序存储，即最近读取的数据放在最前面，最早读取的数据放在最后面，然后它还有一个判断是否删除最老数据的方法，默认是返回false，即不删除数据，我们使用LinkedHashMap实现LRU缓存的方法就是对LinkedHashMap实现简单的扩展，扩展方式有两种，一种是inheritance，一种是delegation，具体使用什么方式看个人喜好

```
//LinkedHashMap的一个构造函数，当参数accessOrder为true时，即会按照访问顺序排序，最近访问的放在最前，最早访问的放在后面
public LinkedHashMap(int initialCapacity, float loadFactor, boolean accessOrder) {
        super(initialCapacity, loadFactor);
        this.accessOrder = accessOrder;
}

//LinkedHashMap自带的判断是否删除最老的元素方法，默认返回false，即不删除老数据
//我们要做的就是重写这个方法，当满足一定条件时删除老数据
protected boolean removeEldestEntry(Map.Entry<K,V> eldest) {
        return false;
}
```

LRU缓存LinkedHashMap(inheritance)实现

采用inheritance方式实现比较简单，而且实现了Map接口，在多线程环境使用时可以使用 Collections.synchronizedMap()方法实现线程安全操作

```
package cn.lzrabbit.structure.lru;

import java.util.LinkedHashMap;
import java.util.Map;

/**
 * Created by liuzhao on 14-5-15.
 */
public class LRUCache2<K, V> extends LinkedHashMap<K, V> {
    private final int MAX_CACHE_SIZE;

    public LRUCache2(int cacheSize) {
        super((int) Math.ceil(cacheSize / 0.75) + 1, 0.75f, true);
        MAX_CACHE_SIZE = cacheSize;
    }

    @Override
    protected boolean removeEldestEntry(Map.Entry eldest) {
        return size() > MAX_CACHE_SIZE;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        for (Map.Entry<K, V> entry : entrySet()) {
            sb.append(String.format("%s:%s ", entry.getKey(), entry.getValue()));
        }
        return sb.toString();
    }
}
```

这样算是比较标准的实现吧，实际使用中这样写还是有些繁琐，更实用的方法时像下面这样写，省去了单独见一个类的麻烦

```
final int cacheSize = 100;
Map<String, String> map = new LinkedHashMap<String, String>((int) Math.ceil(cacheSize / 0.75f) + 1, 0.75f, true) {
    @Override
    protected boolean removeEldestEntry(Map.Entry<String, String> eldest) {
    return size() > cacheSize;
    }
};
```

LRU缓存LinkedHashMap(delegation)实现

delegation方式实现更加优雅一些，但是由于没有实现Map接口，所以线程同步就需要自己搞定了

```
package cn.lzrabbit.structure.lru;

import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Set;

/**
 * Created by liuzhao on 14-5-13.
 */
public class LRUCache3<K, V> {

    private final int MAX_CACHE_SIZE;
    private final float DEFAULT_LOAD_FACTOR = 0.75f;
    LinkedHashMap<K, V> map;

    public LRUCache3(int cacheSize) {
        MAX_CACHE_SIZE = cacheSize;
        //根据cacheSize和加载因子计算hashmap的capactiy，+1确保当达到cacheSize上限时不会触发hashmap的扩容，
        int capacity = (int) Math.ceil(MAX_CACHE_SIZE / DEFAULT_LOAD_FACTOR) + 1;
        map = new LinkedHashMap(capacity, DEFAULT_LOAD_FACTOR, true) {
            @Override
            protected boolean removeEldestEntry(Map.Entry eldest) {
                return size() > MAX_CACHE_SIZE;
            }
        };
    }

    public synchronized void put(K key, V value) {
        map.put(key, value);
    }

    public synchronized V get(K key) {
        return map.get(key);
    }

    public synchronized void remove(K key) {
        map.remove(key);
    }

    public synchronized Set<Map.Entry<K, V>> getAll() {
        return map.entrySet();
    }

    public synchronized int size() {
        return map.size();
    }

    public synchronized void clear() {
        map.clear();
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        for (Map.Entry entry : map.entrySet()) {
            sb.append(String.format("%s:%s ", entry.getKey(), entry.getValue()));
        }
        return sb.toString();
    }
}
```

LRU Cache的链表+HashMap实现

注：此实现为非线程安全，若在多线程环境下使用需要在相关方法上添加synchronized以实现线程安全操作

```
package cn.lzrabbit.structure.lru;


import java.util.HashMap;

/**
 * Created by liuzhao on 14-5-12.
 */
public class LRUCache1<K, V> {

    private final int MAX_CACHE_SIZE;
    private Entry first;
    private Entry last;

    private HashMap<K, Entry<K, V>> hashMap;

    public LRUCache1(int cacheSize) {
        MAX_CACHE_SIZE = cacheSize;
        hashMap = new HashMap<K, Entry<K, V>>();
    }

    public void put(K key, V value) {
        Entry entry = getEntry(key);
        if (entry == null) {
            if (hashMap.size() >= MAX_CACHE_SIZE) {
                hashMap.remove(last.key);
                removeLast();
            }
            entry = new Entry();
            entry.key = key;
        }
        entry.value = value;
        moveToFirst(entry);
        hashMap.put(key, entry);
    }

    public V get(K key) {
        Entry<K, V> entry = getEntry(key);
        if (entry == null) return null;
        moveToFirst(entry);
        return entry.value;
    }

    public void remove(K key) {
        Entry entry = getEntry(key);
        if (entry != null) {
            if (entry.pre != null) entry.pre.next = entry.next;
            if (entry.next != null) entry.next.pre = entry.pre;
            if (entry == first) first = entry.next;
            if (entry == last) last = entry.pre;
        }
        hashMap.remove(key);
    }

    private void moveToFirst(Entry entry) {
        if (entry == first) return;
        if (entry.pre != null) entry.pre.next = entry.next;
        if (entry.next != null) entry.next.pre = entry.pre;
        if (entry == last) last = last.pre;

        if (first == null || last == null) {
            first = last = entry;
            return;
        }

        entry.next = first;
        first.pre = entry;
        first = entry;
        entry.pre = null;
    }

    private void removeLast() {
        if (last != null) {
            last = last.pre;
            if (last == null) first = null;
            else last.next = null;
        }
    }


    private Entry<K, V> getEntry(K key) {
        return hashMap.get(key);
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        Entry entry = first;
        while (entry != null) {
            sb.append(String.format("%s:%s ", entry.key, entry.value));
            entry = entry.next;
        }
        return sb.toString();
    }

    class Entry<K, V> {
        public Entry pre;
        public Entry next;
        public K key;
        public V value;
    }
}
```

LinkedHashMap的FIFO实现

FIFO是First Input First Output的缩写，也就是常说的先入先出，默认情况下LinkedHashMap就是按照添加顺序保存，我们只需重写下removeEldestEntry方法即可轻松实现一个FIFO缓存，简化版的实现代码如下

```
final int cacheSize = 5;
LinkedHashMap<Integer, String> lru = new LinkedHashMap<Integer, String>() {
    @Override
    protected boolean removeEldestEntry(Map.Entry<Integer, String> eldest) {
    return size() > cacheSize;
    }
};
```

调用示例

测试代码

```
package cn.lzrabbit.structure.lru;

import cn.lzrabbit.ITest;

import java.util.LinkedHashMap;
import java.util.Map;

/**
 * Created by liuzhao on 14-5-15.
 */
public class LRUCacheTest  {

    public static void main(String[] args) throws Exception {
        System.out.println("start...");

        lruCache1();
        lruCache2();
        lruCache3();
        lruCache4();

        System.out.println("over...");
    }


 static   void lruCache1() {
        System.out.println();
        System.out.println("===========================LRU 链表实现===========================");
        LRUCache1<Integer, String> lru = new LRUCache1(5);
        lru.put(1, "11");
        lru.put(2, "11");
        lru.put(3, "11");
        lru.put(4, "11");
        lru.put(5, "11");
        System.out.println(lru.toString());
        lru.put(6, "66");
        lru.get(2);
        lru.put(7, "77");
        lru.get(4);
        System.out.println(lru.toString());
        System.out.println();
    }


static   <T> void lruCache2() {
        System.out.println();
        System.out.println("===========================LRU LinkedHashMap(inheritance)实现===========================");
        LRUCache2<Integer, String> lru = new LRUCache2(5);
        lru.put(1, "11");
        lru.put(2, "11");
        lru.put(3, "11");
        lru.put(4, "11");
        lru.put(5, "11");
        System.out.println(lru.toString());
        lru.put(6, "66");
        lru.get(2);
        lru.put(7, "77");
        lru.get(4);
        System.out.println(lru.toString());
        System.out.println();
    }

  static  void lruCache3() {
        System.out.println();
        System.out.println("===========================LRU LinkedHashMap(delegation)实现===========================");
        LRUCache3<Integer, String> lru = new LRUCache3(5);
        lru.put(1, "11");
        lru.put(2, "11");
        lru.put(3, "11");
        lru.put(4, "11");
        lru.put(5, "11");
        System.out.println(lru.toString());
        lru.put(6, "66");
        lru.get(2);
        lru.put(7, "77");
        lru.get(4);
        System.out.println(lru.toString());
        System.out.println();
    }

  static  void lruCache4() {
        System.out.println();
        System.out.println("===========================FIFO LinkedHashMap默认实现===========================");
        final int cacheSize = 5;
        LinkedHashMap<Integer, String> lru = new LinkedHashMap<Integer, String>() {
            @Override
            protected boolean removeEldestEntry(Map.Entry<Integer, String> eldest) {
                return size() > cacheSize;
            }
        };
        lru.put(1, "11");
        lru.put(2, "11");
        lru.put(3, "11");
        lru.put(4, "11");
        lru.put(5, "11");
        System.out.println(lru.toString());
        lru.put(6, "66");
        lru.get(2);
        lru.put(7, "77");
        lru.get(4);
        System.out.println(lru.toString());
        System.out.println();
    }

}
```

运行结果

```
"C:\Program Files (x86)\Java\jdk1.6.0_10\bin\java" -Didea.launcher.port=7535 "-Didea.launcher.bin.path=C:\Program Files (x86)\JetBrains\IntelliJ IDEA 13.0.2\bin" -Dfile.encoding=UTF-8 -classpath "C:\Program Files (x86)\Java\jdk1.6.0_10\jre\lib\charsets.jar;C:\Program Files (x86)\Java\jdk1.6.0_10\jre\lib\deploy.jar;C:\Program Files (x86)\Java\jdk1.6.0_10\jre\lib\javaws.jar;C:\Program Files (x86)\Java\jdk1.6.0_10\jre\lib\jce.jar;C:\Program Files (x86)\Java\jdk1.6.0_10\jre\lib\jsse.jar;C:\Program Files (x86)\Java\jdk1.6.0_10\jre\lib\management-agent.jar;C:\Program Files (x86)\Java\jdk1.6.0_10\jre\lib\plugin.jar;C:\Program Files (x86)\Java\jdk1.6.0_10\jre\lib\resources.jar;C:\Program Files (x86)\Java\jdk1.6.0_10\jre\lib\rt.jar;C:\Program Files (x86)\Java\jdk1.6.0_10\jre\lib\ext\dnsns.jar;C:\Program Files (x86)\Java\jdk1.6.0_10\jre\lib\ext\localedata.jar;C:\Program Files (x86)\Java\jdk1.6.0_10\jre\lib\ext\sunjce_provider.jar;C:\Program Files (x86)\Java\jdk1.6.0_10\jre\lib\ext\sunmscapi.jar;C:\Program Files (x86)\Java\jdk1.6.0_10\jre\lib\ext\sunpkcs11.jar;D:\SVN\projects\Java\Java.Algorithm\target\test-classes;D:\SVN\projects\Java\Java.Algorithm\target\classes;C:\Program Files (x86)\JetBrains\IntelliJ IDEA 13.0.2\lib\idea_rt.jar" com.intellij.rt.execution.application.AppMain Main
start... ===========================LRU 链表实现=========================== 5:11 4:11 3:11 2:11 1:11 4:11 7:77 2:11 6:66 5:11 ===========================LRU LinkedHashMap(inheritance)实现=========================== 1:11 2:11 3:11 4:11 5:11 5:11 6:66 2:11 7:77 4:11
===========================LRU LinkedHashMap(delegation)实现=========================== 1:11 2:11 3:11 4:11 5:11 5:11 6:66 2:11 7:77 4:11 ===========================FIFO LinkedHashMap默认实现=========================== {1=11, 2=11, 3=11, 4=11, 5=11} {3=11, 4=11, 5=11, 6=66,
7=77} over... Process finished with exit code 0
```