* 学习目标

  * 能够掌握Map子类LinkedHashMap

  * 能够掌握Map子类TreeMap

  * 能够理解Set接口概述

  * 能够通过阅读HashSet源码理解它底层实现就是HashMap的key集合

  * 能够通过阅读LinkedHashSet源码理解它底层实现就是LinkedHashMap的key集合

  * 能够通过阅读TreeSet源码理解它底层实现就是TreeMap的key集合

----

  * 能够掌握Map子类LinkedHashMap

    * 概述：

      * LinkedHashMap是继承于HashMap，是基于HashMap和双向链表来实现的

      * HashMap无序而LinkedHashMap有序，可分为插入顺序和访问顺序两种

      * LinkedHashMap存取数据，还是跟HashMap一样使用的Entry[]的方式，双向链表只是为了保证顺序

      * LinkedHashMap是线程不安全的

```
* HashMap是存放无序而LinkedHashMap是存放有序的（插入顺序）
public static void main(String[] args) {
        Map<String,String> map1=new HashMap<String,String>();
        map1.put("name", "xiaohei");
        map1.put("age", "18");
        map1.put("sex", "男");
        // 存放是无序
        Set<Entry<String, String>> entries = map1.entrySet();
        for(Entry<String, String> entry:entries) {
            System.out.printf("%s %s ",entry.getKey(),entry.getValue());
        }
        System.out.println();
        Map<String,String> map2=new LinkedHashMap<String,String>();
        map2.put("name", "xiaohei");
        map2.put("age", "18");
        map2.put("sex", "男");
        // 存放是有序
        entries = map2.entrySet();
        for(Entry<String, String> entry:entries) {
            System.out.printf("%s %s ",entry.getKey(),entry.getValue());
        }
    }
 代码结果：
 sex 男 name xiaohei age 18
 name xiaohei age 18 sex 男

 * 访问顺序
    public static void main(String[] args) {
        Map<String,String> map=new LinkedHashMap<String,String>(16, 0.75f, true
        for (int i = 0; i < 10; i++) {
            map.put(String.valueOf(i), String.valueOf(i));
        }
        // 访问置顶
        map.get("6");
        Set<Entry<String, String>> entries = map.entrySet();
        for(Entry<String, String> entry:entries) {
            System.out.printf("%s ",entry.getValue());
        }
    }
 结果：
    0 1 2 3 4 5 7 8 9 6
 备注：  通过例子可以看出，最近经常使用的元素就放在后面，最近最少使用的就排在了链表的前
 可断点调试：查看结果

* 通过LinkedHashMap（顺序访问的特点）
 * 编写简单一个LRU（Least Recently Used）最近最少使用缓存
import java.util.LinkedHashMap;
public class LruCache<K,V> extends LinkedHashMap<K, V> {
    private static final long serialVersionUID = 14353245324L;
    private int maxSize;// 最大可以缓存多少个
    public LruCache(int maxSize){
        super(16, 0.75f, true);
        this.maxSize=maxSize;
    }
    //重写删除最老元素的方法
    @Override
    protected boolean removeEldestEntry(java.util.Map.Entry<K, V> eldest) {
        // 当集合的大小超过缓存最大的值，删除最老的元素
        return size()>maxSize;
    }
}

public static void main(String[] args) {
        LruCache<String, String> lruCache=new LruCache<String,String>(8);
        for (int i = 0; i < 10; i++) {
            lruCache.put(String.valueOf(i), String.valueOf(i));
        }
        lruCache.get("5");
        Set<Entry<String, String>> entries = lruCache.entrySet();
        for(Entry<String, String> entry:entries) {
            System.out.printf("%s ",entry.getValue());
        }

}
结果：
    2 3 4 6 7 8 9 5
断点调试
* 分析removeEldestEntry源码调用过程
  *  void afterNodeInsertion(boolean evict) { // possibly remove eldest
        LinkedHashMap.Entry<K,V> first;
        if (evict && (first = head) != null && removeEldestEntry(first)) {
            K key = first.key;
            removeNode(hash(key), key, null, false, true);
        }
    }
  * afterNodeInsertion 这个方法是put的的调用到

 * LruCache案例的编写
    * 提前感受一下IO流和网络编程
 public static void main(String[] args) throws Exception {
        // https://www.jd.com:e1d84f4301e444a3db82c908f29947b1
        // https://www.taobao.com:1b263ad2261782bddac1d35c0528d40e
```

```java
        // https://www.baidu.com:f9751de431104b125f48dd79cc55822a
        LruCache<String, String> cache=new LruCache<String,String>(2);
        Scanner input=new Scanner(System.in);
        String path="";
        System.out.println("请输入url的地址(推出请输入n): ");
        path=input.next();
        while(!("n".equals(path))) {
            String key=MD5Utils.md5(path);
            String content=null;
            if(cache.containsKey(key)) {
                // 缓存中已经存在，直接从缓存中
                content = cache.get(key);
                System.out.println(content);
                System.out.println("从缓存中获取");
            }else {
                //从网络获去内容
                content = getNetContent(path);
                cache.put(key, content);
                System.out.println(content);
                System.out.println("从网络中获取");
            }
            Set<Entry<String, String>> entries = cache.entrySet();
            for(Entry entry:entries) {
                System.out.printf("key=%s ",entry.getKey());
            }
            System.out.println();
            System.out.println("请输入url的地址(推出请输入n): ");
            path=input.next();
        }
        System.out.println("程序结束...");
    }

    private static String getNetContent(String path) throws MalformedURLExcepti
        // 构建链接
        URL url=new URL(path);
        HttpURLConnection connection = (HttpURLConnection) url.openConnection(
        // 链接起来
        connection.connect();
        // 获得链接的输入流
        InputStream is = connection.getInputStream();
        // 把输入流转换成字符成
        byte[] buffer=new byte[1024];
        int len=0;
        StringBuilder sb=new StringBuilder();
        while((len=is.read(buffer))!=-1) {
            String value=new String(buffer,0,len,Charset.forName("UTF-8"));
            sb.append(value);
        }
        String content=sb.toString();
        is.close();
        return content;
    }

```

* 能够掌握Map子类TreeMap

  * TreeMap概述

    * 底层用黑红树实现的

    * TreeMap中的元素默认按照keys的自然排序排列

    * 使用TreeMap前提

      * 要么Key的类实现了Comparable接口

      * 假如Key的类没有实现Comparable接口，就需要使用Comparator比较器

```
* Integer: 数字的升序, 已经实现了Comparable接口
* String: 按照字母表排序, 已经实现了Comparable接口
public static void main(String[] args) {
        // Integer
        Map<Integer,String> treeMap=new TreeMap<Integer,String>();
        treeMap.put(2, "关羽");
        treeMap.put(1, "刘备");
        treeMap.put(3, "张飞");
        Set<Entry<Integer, String>> entries = treeMap.entrySet();
        for(Entry<Integer, String> entry:entries) {
            System.out.printf("%s %s ",entry.getKey(),entry.getValue());
        }
        System.out.println();
        // String
        Map<String,String> treeMap1=new TreeMap<String,String>();
        treeMap1.put("B", "关羽");
        treeMap1.put("A", "刘备");
        treeMap1.put("C", "张飞");
        Set<Entry<String, String>> entries1 = treeMap1.entrySet();
        for(Entry<String, String> entry:entries1) {
            System.out.printf("%s %s ",entry.getKey(),entry.getValue());
        }
    }
结果:
 1 刘备 2 关羽 3 张飞
 A 刘备 B 关羽 C 张飞

 * 根据User的Id排序购物车
 public class User {
    private int id;
    private String name;
    private String password;
    ...
}
public class Goods {
    private int id;
    private String name;
    private double price;
    ...
}
public class Cart {
    private int userId;
    private List<Goods> cartList;
    ...
}
    * HashMap写法
    public static void main(String[] args) {
        Map<User,Cart> carts=new HashMap<User,Cart>();
        carts.put(new User(1001,"刘备","123"), new Cart());
        carts.put(new User(1002,"关羽","123"), new Cart());
        carts.put(new User(1003,"张飞","123"), new Cart());
        Set<Entry<User, Cart>> entries = carts.entrySet();
        for(Entry<User, Cart> entry:entries) {
            System.out.println(entry.getKey());
        }
    }
  结果(无序):
  User [id=1002, name=关羽, password=123]
  User [id=1001, name=刘备, password=123]
  User [id=1003, name=张飞, password=123]
* TreeMap: 要么User实现Comparable,要么使用Comparator构造器
  * 假如两种都不使用会报异常: xx cannot be cast to java.lang.Comparable
  * 第一种实现Comparable接口
   public class User implements Comparable<User>
   @Override
   public int compareTo(User o) {
       return id-o.id;
   }
   public static void main(String[] args) {
       Map<User,Cart> carts=new TreeMap<User,Cart>();
       carts.put(new User(1001,"刘备","123"), new Cart());
       carts.put(new User(1002,"关羽","123"), new Cart());
       carts.put(new User(1003,"张飞","123"), new Cart());
       Set<Entry<User, Cart>> entries = carts.entrySet();
       for(Entry<User, Cart> entry:entries) {
           System.out.println(entry.getKey());
       }
   }
结果:
User [id=1001, name=刘备, password=123]
User [id=1002, name=关羽, password=123]
User [id=1003, name=张飞, password=123]
 * 第二种使用Comparator
   public static void main(String[] args) {
       Map<User,Cart> carts=new TreeMap<User,Cart>(new Comparator<User>() {

           @Override
           public int compare(User o1, User o2) {
               return o1.getId()-o2.getId();
           }
       });
       carts.put(new User(1001,"刘备","123"), new Cart());
```

```
        carts.put(new User(1002,"关羽","123"), new Cart());
        carts.put(new User(1003,"张飞","123"), new Cart());
        Set<Entry<User, Cart>> entries = carts.entrySet();
        for(Entry<User, Cart> entry:entries) {
            System.out.println(entry.getKey());
        }
    }
 * 结果:
    User [id=1001, name=刘备, password=123]
    User [id=1002, name=关羽, password=123]
    User [id=1003, name=张飞, password=123]
 * 假如既实现Comparable接口，又使用了Comparator,Comparator的优先级比较高
```

* 能够理解Set接口概述

* 能够通过阅读HashSet源码理解它底层实现就是HashMap的key集合

* 能够通过阅读LinkedHashSet源码理解它底层实现就是LinkedHashMap的key集合

* 能够通过阅读TreeSet源码理解它底层实现就是TreeMap的key集合