集合

- **特点**
  - 与数组比较，数组长度开始的时候必须指定，且一旦指定，不能随意更改
  - 保存的必须为统一类型的元素，集合可以保存所有类型
  - 使用数组进行增加删除元素，比较麻烦，（扩容，分配空间）
- **集合的框架体系**
  - 分为两大类
    - 一类（List，Set实现的是单例集合） — Iterable — Collection
      - List
        - linkedList
        - ArrayList
        - Vector
      - Set
        - TreeSet
        - HashSet
    - 二类（Map接口实现的是双列集合（键值对形式）） — Map
      - HashMap — LinkedHashMap
      - Hashtable — Properties
      - TreeMap
- **Collection接口和常用方法**
  - Collection接口实现类的特点
    - 实现子类可以存放多个元素，每个元素可以是Object
    - 有些实现类可以存放重复的元素，有些不可以
    - 有些实现类是有序的LIst，有些是无序的（Set）
    - Collection没有直接实现子类，是通过其子接口Set和List实现的
  - 常用方法（以Arrylist来演示）（子类必须实现接口的方法）
    - add — 可以添加数字字符串boolean类型，自动装箱
    - addAll — 添加多个元素，通常是添加一个新的集合进去 — 传一个集合进去
    - remove — 删除指定元素 — 可以删除索引在的元素，也可以指定元素删除
    - remove — 删除多个元素（重载） — 传一个集合进去
    - contains — 查找元素是否存在，返回Boolean值
    - containsAll — 查找多个元素是否都存在 — 传一个集合进去
    - size — 获取元素个数
    - isEmpty — 判断是否为空
    - clear — 清空数组
  - 接口遍历元素的方式（Arrylist和linkedList可以使用）
    - 迭代器
      - 所有实现了Collection接口的集合类都有一个iterator（）方法，用于返回一个实现了Iterator接口的对象，即一个迭代器
      - 仅用于遍历集合
      - 迭代器方法
        - hasNext（）判断是否还有下一个元素
        - next（）作用：1.下移 2，将下移后的元素返回
        - 注意：在调用迭代器next方法前必须调用hasnext方法，如果不调用，则下一个记录无效，直接调用next会抛出nosuchElemntException异常
      - 使用
        - 先得到集合的迭代器通过iterator（）方法，然后使用while循环进行遍历 — 快捷方式itit — ctrk+j返回所有快捷提示
        - 退出while循环后，这时迭代器指向最后的元素 — 如果需要再次遍历，需要重置迭代器，即重新返回一个迭代器给上一个迭代器对象接受者
    - 增强for循环
      - 本质还是迭代器，也存在使用完后迭代器的游标在最后的情况
      - 运用范围更广泛，可用于集合数组
    - 普通for循环搭配size方法和get方法
  - List接口和常用方法（都是有序的）
    - 特点
      - List集合中的元素有序，添加和取出顺序一致，且可以重复
      - 每个元素都有其对应的顺序索引，即支持索引
      - List接口的实现类有很多
      - list容器的元素都对应一个整数型的序号记载在容器中的位置，可以根据序号存取容器中的元素
    - 常用方法（以Arrylist为例子）
      - add — 有重载方法，即先确定在那个位置添加，后面传入添加的元素
      - addAll — 传入一个index，传入一个集合，整个添加进去
      - remove — 移除指定位置的元素，返回此元素/或者传入要确定删除的元素
      - removeall — 删除多个元素
      - set — 传入一个指定的位置，将那个位置的元素改为传入的元素（相当于替换）
      - contains — 查找元素是否存在
      - containsall — 查找多个元素是否都存在
      - get — 传入索引得到元素
      - indexOf — 返回查找对象在集合中首次出现的位置
      - lastindexOf — 返回查找对象字集合中最后出现的位置
      - subList — 返回大小集合，相当于在集合中分割一段返回，左闭右开
      - isempty — 判断是否为空
      - size — 返回数组的大小
      - clear — 清空数组
    - ArrayList底层结构和源码分析（相当于一个自动的数组，效率高但是线程不安全）
      - 注意事项
        - 可以加入任何元素，且可以添加相同的元素，也可以添加null，且有多个
        - ArrayList是通过数组来实现数据存储的
        - ArrayList基本等同于Vector，执行效率高但是不安全，在多线程情况下不建议用ArrayList
      - 源码分析
        - ArrayList中维护了一个object类型的数组（故返回的都是object类型）
        - 当创建ArrayList对象时，如果使用的是无参构造器，则初始容量为0，第一次添加，则扩容为10，如果需要再次扩容，则需要扩容1.5倍
        - 如果使用的是指定大小的构造器，初始数组指定为指定大小，如果需要扩容，直接扩容到1.5倍
    - 互为补充
    - Vector底层源码分析（相当于一个自动数组，效率低，但是线程安全）
      - 说明
        - 底层也是一个对象数组（返回对象也是object类型）
        - 他是线程同步的，即线程安全，操作方法带有synchronized
        - 在开发中，需要线程同步安全时，考虑使用Vector
      - 源码分析
        - 如果是默认构造器，默认10，满后就按照2倍扩容
        - 如果是指定扩容，则每次扩容直接按照二倍进行扩容
    - 互为补充
    - LinkedList底层源码分析
      - 底层实现了双向链表和双端队列（不是通过数组来完成的，效率更高）
      - 可以添加任意元素，元素可以重复，包括null
      - 线程不安全，没有实现同步
      - 如果增删操作多，可以考虑选择linkedlist，但是大部分情况下查询较多，所以一般选择ArrayList
      - 底层操作机制
        - 底层维护了一个双向链表
        - LinkList中维护了两个属性first和last分别指向收节点和尾节点
        - 每个节点（Node对象），里面又维护了prev，next，item三个属性，其中通过prev指向前一个节点，next指向后一个节点，最终实现双向链表
        - LinkList的元素的添加和删除，不是通过数组完成的，所以效率相对较高（但是改查效率低）
        - 增删数据就是改变里面的指向关系，更加方便
      - 方法
        - add
        - remove，默认删除第一个，可以指定删除
        - get 得到对应节点的内容
        - set（想要修改的节点的索引，修改的内容）
  - set接口和常用方法
    - 特点
      - 无序（添加和取出顺序不一致（但是hash后的顺序就固定了），没有索引）
      - 不允许重复元素，所以最多包含一个null
      - 遍历方式有迭代器和增强for两种，因为没有索引，索引不可以使用常规的for循环实现
    - 常用方法（也是Collection接口的实现）（子类必须实现接口的方法）
      - add — 可以添加数字字符串boolean类型，自动装箱
      - addAll — 添加多个元素，通常是添加一个新的集合进去 — 传一个集合进去
      - remove — 删除指定元素（指定到对象） — 可以删除索引在的元素，也可以指定元素删除
      - contains — 查找元素是否存在，返回Boolean值
      - containsAll — 查找多个元素是否都存在 — 传一个集合进去
      - size — 获取元素个数
      - isEmpty — 判断是否为空
      - clear — 清空数组
    - 接口实现类
      - HashSet — 实际上是底层是HashMap，hashmap的底层是数组加链表（单向）加红黑树 — Hashmap底层是数组加链表加红黑树
      - 子类
      - 底层源码解读
        - 先获取元素的hash值，hashcode（）方法
        - 对哈希值进行计算得出索引值即为要存放在hash表中的位置号
        - 如果该位置上没有其他元素，则直接存放，如果有，则要进行equals判断，如果相等，则不添加，如果不相等，则以链表的形式添加
        - 如果一条链表的元素个数达到8，并且table大小大于等于64，则会进行树化，否则会继续扩容 — 树化机制
        - HashSet底层是hashmap，第一次添加时，table数组扩容到16，临界值（threhold）是16*0.75=12
        - 如果table数组使用到了临界值12，就会扩容到16*2=32，新的临界值就是32*0.75=24，依次类推 — 扩容机制
      - 补充，使之有序
      - linledHashset
        - 是Hashset的子类，底层是LinkedHashMap，底层维护了一个数组加双向链表（该类中有属性head和tail）
        - LinkedHashset根据元素的hashcode值决定元素的存储位置先算hash值在求索引，确定元素在table表的位置，然后将添加的元素加入双向链表，如果已经存在，就不添加，同时使用链表维护元素的次序，这使得元素看起来是以插入顺序保存的
        - 不准许添加重复元素
        - 每一个节点有before和after元素，这样可以形成双向链表
        - 可以保证插入顺序和遍历顺序一致
      - Treeset
        - 主要特点，可以排序
        - 使用
          - 当我们使用无参构造器的时候，创建treeset的时候，是无序的
          - 可以传入比较器对象(匿名内部类)，进行定制排序（一般均需要向下转型），比较的到时0的话该元素不会被添加
  - Map接口与常用方法
    - 特点
      - 用于保存k-v数据，且k,v可以为任意类型
      - key不可以重复，v可以重复，因为要确定唯一key找到值（如果key重复了，会得到相同的hash值，然后对前面的相同key的数据进行替换）
      - k可以为null，v也可以为null，但是k只能有一个为null
      - 常用String类作为key
      - 数据存放结构 — k-v存放在hash$Node中，Entryset存放Entry类型的数据，Node实现了Entry接口，故存放在Entryset里面
      - k,v一一对应，通过k总是可以找到v
    - 取出所有的值
      - 使用values方法，在使用增强for循环
      - 使用迭代器
    - map接口常用方法
      - put（k，v）----增加元素
      - remove----根据键删除映射关系
      - get-----根据键获取value
      - isempty-----判读个数是否为0
      - size----获取元素个数
      - clear-----清空元素
      - containsKey----判断某个键值是否存在
      - keyset（）获取所有的键，返回的是set类型
      - entryset----获取所有关系的k-v — 返回Collection类型
      - values----获取所有的值，，返回Collection类型
    - map接口遍历方式
      - 增强for循环 — 先使用keyset（）取出所有的key值，for循环，在根据get方法取出values
      - 迭代器 — 先使用keyset（）取出所有的key值，在调用他的迭代器，在配合get方法取出values
      - 通过entryset来获取k-v
        - 使用增强for，然后再强转成Map.Entry对象，在调用该对象的getkey和getvalues方法
        - 使用迭代器，然后再强转成Map.Entry对象，在调用该对象的getkey和getvalues方法
    - Map接口实现类
      - HashMap
        - 小结
          - Map接口的常用实现类：HashMap，Hashtable，Properties
          - HashMap是以Map接口使用频率最高的实现类
          - HashMap是以键值对的方式来储存数据
          - key不能重复，但是值可以重复，允许使用null键和null值
          - 如果添加相同的key，则会覆盖原来的k，v等同于修改（key不会替换，val可以替换）
          - 与Hashset一样，不保证映射的顺序，因为底层是以hash表的方式来存储的 — 底层 数组加链表加红黑树
          - Hash没有实现同步，因此是线程不安全的方法没有做同步互斥的操作，没有synchronized
        - 底层源码分析
          - 维护了Node类型的数组table，默认为null
          - 当创建对象时将加载因子（LOadfactor）初始化为0.75
          - 当添加k-v时，通过k的hash值得到在table的索引，然后判断该索引处是否有元素，如果没有，则添加，如有元素，则判断该元素的key和准备加入的key是否相等，如果相等，则替换，如果不相等风需要判断是树结构还是链表结构，做出相应的处理，如添加时发现容量不够，则需要扩容
          - 第一次添加，则需要扩容table的容量为16，临界值为12
          - 以后在扩容，则需要扩容的容量为原来的两倍，临界值也为原来的两倍、
          - 如果一条链表的元素超过8，且容量大于64，就会进行树化
      - 补充
      - Hashtable（版本更久，效率低，不能添加null键和null值）
        - 介绍
          - 存放的元素是键值对
          - 键和值都不能为空，否则会抛出空指针异常
          - 使用方法基本和HashMap一样
          - 他是线程安全的
      - 子类
      - Properties
        - 介绍
          - 继承Hashtable并实现了Map接口，也用键值对的形式来保存数据
          - 使用和Hashtable类似
          - 可用于从xxx.properties文件中，加载数据到Properties类对象，并进行读取和修改
          - 说明:工作后XXX.properties文件通常作为配置文件 — 保存密码
        - 基本使用
          - 键值均不能为空
          - 常用方法
            - put（可用于替换）
            - get，通过键来获取元素
            - remove，通过键来删除元素
      - treemap
        - 主要特点，可以定制排序（对key）
        - 使用
          - 使用默认的构造器，创建TreeMap，是无序的
          - 可以传入比较器对象(匿名内部类)，进行定制排序（一般均需要向下转型），比较的到时0的话该元素不会被添加 — 搭配泛型，可以避免转型
          - 里面保存的是Entry类型的数据

开发中如何选择集合类

- 先判断存储的类型，是单列还是双列
  - 单列 — Collections接口
    - 允许重复 — list
      - 改查多 — ArrayList
      - 增删多 — LinkedLIst
    - 不允许重复 — set
      - 无序 — hashset
      - 排序 — treeset搭配构造器可以定制排序
      - 插入和取出顺序一致 — linkedhashset
  - 双列 — Map接口
    - 键无序 — Hashmap
    - 键排序 — treemap，搭配构造器可以定制排序
    - 插入和取出顺序一致 — linkedhashmap
    - 读取文件 — properties

Collections工具类

- 介绍
  - 用于操作set list map等集合的工具类
  - 提供了一系列静态的方法处理集合
- 常用方法
  - reverse — 翻转元素的顺序
  - shuffle — 对list集合元素进行随机排序
  - sort
    - 根据自然顺序对指定的list集合元素按升序排序 — 首字母的ascii码
    - 传入比较器定制排序
  - swap — 交换两个地方的元素 — swap（list，i，j）
  - max
    - 根据元素的自然排序，返回给定集合中最大的元素
    - 也可以定制，可以实现定制排序，可以实现min（）的效果
  - min — 同max
  - frequency — 统计集合中指定元素出现的次数
  - copy — copy（目标数组，原数组） — 注意：目标数组要和原数组的大小一致，可以利用for循环加size方法配合写入空的方式来完成这个条件
  - replaceall — 替换所有的指定的元素