# 基本类型与集合疑问点

基本类型与对象：

|  |
| --- |
| byte(8位)｛-2^7，2^7-1｝,即10000000-0111111，-128到127  为什么是2^7-1?  眼高手低！ 1111111 在十进制就是127=27-1  这跟数学有关。 三位数，最大是 111，二进制为 1+2^1+2^2=7 , 即 23-1.  但是23 有八种可能，其实是0占了一位，导致了-1.  所以28 的范围是【-127，127】，首位是符号位，但是+0-0重复，便规定-0=-128 |
| Int 转 float 会精度丢失?  这里牵扯到float类型的结构， float是32位，1位符号位8位指数位，23位尾数位  虚线转换都会精度缺失。  int n = 123456789;  float f =n; 打印f，发现***f=1.23456792E8***，发生了精度损失。  注意了，**浮点型的表示其实是科学计数法，是带指数的！！ 不是100.00！**  **Int是32位，但如果超过了24位的话，float就没法表示了，因为只有23位尾数+一个整数位，最多表示24位。**  Int 转 浮点型表示，一定要转Double而不是float。 |
| **Arrays函数** equals()：比较两个array是否相等。array拥有相同元素个数，且所有对应元素两两相等。 fill()：将值填入array中。Arrays.fill(arr, 10); //将数组arr全部填充10 sort()：用来对array进行排序。 binarySearch()：在排好序的array中寻找元素。 System.arraycopy()：array的复制。 |
| 迭代器的写法  Iterator<String> it = list.iterator();  while(it.hasNext()){  String x = it.next();  if(x.equals("del")){  it.remove();  }  } |
| **Comparator接口**  【一句话 ：返回负数,第一个参数放前面; 返回0和1 不变。】  **Collections.sort( list** , new Comparator<Student>(){  @Override  public int compare(Student o1, Student o2) {  if (o1.age > o2.age) {  return 1;  } else if (o1.age < o2.age) {  return -1; 即 升序。  }  return 0;  }  });  **排序其实是由三个数字同时决定的；**  升序（默认，即官方定义，毕竟代码实现就是基于这个写的）：  *< -1*  *= 0 //或者 1效果是一样的；-1相同元素会发生位置调整*  *> 1*  降序：  < 1  = 0 //或者 1效果是一样的；-1相同元素会发生顺序调整  > -1  倒序：  //直接  return -1;  不改变顺序：  //直接  return 0或者1; |
| Comparable 可比较的。 这个接口只有一个方法 compareTo  实现这个接口的Student直接可以 Collecion.sort(stuList)  Comparator 比较器。 这个接口允许不改变源码就可比较。重写conpare方法并将自定义类的对象传入即可。  Collections.sort(list,comparator);  static class UserComparator implements Comparator<User> {  @Override  public int compare(User o1, User o2) {  …………….  }  }  也可以用内部类，写法见前文 |
| TreeMap，TreeSet等使用的时候要注意：  如果key没有实现Comparable接口的话必须TreeMap(new Comparator)的方式构建 |