# 阶段4-Java语言进阶【集合】

### 一、上节课回顾

### 二、作业点评

### 三、本次课知识点，重点难点

【知识点目标】

• 泛型

• 集合框架的构成和常用的接口

• 迭代器

• Collection 和 Map

• 比较器

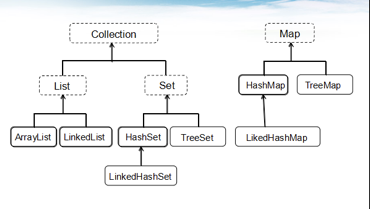
【重点】

• Collection 和 Map 的使用

【难点】

### 四、本次课程任务知识点讲解

集合框架的构成和常用的接口



集合框架体系

由来：数据多了可以用对象类存储，对象多了用集合来存。

理解：咱班就是一个集合，每个人都是单独封装了功能的对象，而且长度是不固定的，因为每天都有人请假，77人或者76人，所以这个集合的长度是可变的。

概述：集合是一个容器、存储对象、长度可变化。

什么时候使用容器：一旦数据多了就得先把数据存储起来再操作，这样效率更好

和数组的区别：

1. 数组定义时长度是固定的(必须明确元素类型和长度) 集合定义时，可以不用明确长度，长度是可变的。

2. 数组存储是同一类型的数据，集合存储的是对象，而对象可以是任意类型

3. 数组可以存储基本数据值，而集合只能对象。

集合框架构成与分类：

数据结构？什么是数据结构？有哪些体现方式？

每一个容器因为自身的数据机构不同，进行了单独的描述和对象的封装，以为这些对象有共性就进行向上抽取，形成了集合框架，

Collection接口的常见方法：

* 增

Add();添加一个元素

AddAll();添加一个集合，把小集合添加到大集合中

* 删

Remove()；删除一个,删除会返回Boolean类型，用到equals方法判断是否相同。如果删除成功，会改变集合的长度，下面同理。

removeAll()；删除一部分，按照指定的集合的元素来删

Clear()；清空

* 判断：

Contains()；包含，先遍历对象的元素，用到了对象判断的equals方法

isEmpty()；是否为空

Size()；集合的元素的大小或者说长度

* 取

retainAll()；取交集

Iterator()；取出元素

* 转变

toArray()；

理解总结：把我们现在的班级想成容器即可，班上来一个新同学add()添加一个，来了一堆同学addAll();添加一部分，班上出去一个remove()删除一个，出去一部分removeAll()删除一部分。班上是不是有人isEmpty()；班上有多少人Size();班上人全部都出去clear()班上是否包含张三contains();和隔壁班有没有相同的学生retainAll();

实现代码：

public static void main(String[] args)

{

Collection coll=new ArrayList();

methodDemo(coll);

}

public static void methodDemo(Collection coll)

{

//添加：

coll.add("abc1");

coll.add("abc2");

coll.add("abc3");

coll.add("abc4");

sop(coll);//输出集合中的元素，

//删除一个

coll.remove("abc2");删除会返回boolean类型。用到equals方法判断是否相同。

sop(coll.size()); size 3

总结：删除如果成功，会改变集合的长度。

//清空

coll.clear();

//判断元素是否存在

boolean b=coll.contains("abc3");先遍历对象元素，用到了对象判断的equals方法。

sop(b);true

coll.isEmpty();判断是否为空。

演示带All的方法 addAll\removeAll\containsAll\retainAll

public static void methodAllDemo(Collection c1,Collection c2)

{

c1.add("abc1");

c1.add("abc2");

c1.add("abc3");

c1.add("abc4");

sop(c1);

c2.add("abc1");

c2.add("abc5");

c2.add("abc6");

sop(c2);

c1.addAll(c2);会出现重复元素，把两个集合中所有的元素都添加到当中。

boolean b=c1.containsAll(c2);c1中是否包含c集合中的元素、

全部都有才算是包含，有一个没有都不行。

boolean b=c1.removeAll(c2);将c1和c2中相同的元素在c1中删除。

当没有相同的元素时，返回false.

boolean b=c1.retianAll(c2);true 取出相同的部分。将c2和c1相同的元素

保存在c1中，不同的元素删除。

注意：只要集合发生了变化就会返回真，不发生变化时返回false

当c1是c2的一个子集的时候就会返回false(),因为c1没有发生变化。

}

}

取出；

interator();

public static void getElements(Collection coll)

{

coll.add("abc1");

coll.add("abc2");

coll.add("abc3");

coll.add("abc4");

Iterator it=coll.iterator();

while(it.hasNext())

{

sop(it.next());

//sop(it.next());不能这样取。

}

迭代器取出对象元素的方式。next方法只能出现一次，避免问题的发生，没有这个元素异常。

因为它只判断一次，但是取出两个元素的话。

sop(it.hasNext());//为假，上面迭代器去完了之后就没用了。

为了优化内存的另外一种写法：因为it对象使用之后并没有指向其他任何引用，任然保存在内存中。

for(Iterator it=coll.iterator();it.hasNext();)

{

sop(it.next());

}

开发中常用！

/\*

sop(it.next());

sop(it.next());

sop(it.next());

sop(it.next());

sop(it.next());//NoSuchElementException:没有这个元素异常。

\*/这种做法不行，体现不出专业性，以及代码的复用性可言。

}

迭代器的设计原理以及思想：

其实就Collection这种集合的公共取出方式，因为每一个具体的容器的数据结构不同，它们的取出方式也是不一样的，要根据每一个容器的特点来定义特定的取出方式，而取出方式直接在访问容器中的元素，所以将取出方式定义在容器内部，是一个内部类，而对外提供Iterator接口，这样可以通过该接口访问任意collection容器中的元素，所以它的出现降低了耦合性，我们只要面对Iterator接口，使用iterator接口即可。

集合使用的细节：

1. 直接存储基本数据类型值也是可以的，因为JDK1.5后有自动装箱，会将基本数据类型转成对象，JDK1.4绝对不行。Eg:coll.add(3)

2. 集合对象中存储的其实是对象的元素的引用。

3. Add的参数是Object类型，可以接收所有的类型的对象，会出现向上转型，取出元素时类型还是Object不能使用具体对象的特有内容，想要使用特有内容向下转型。Eg:Object obj="obj";想要取出访问子类的特有方法，向下转型，String s=(String)it.next();sop(s.length());

collection框架体系：

Collection

* list(列表)：有序（存入的顺序和取出的顺序一致）因为该字体系中的元素有索引或者角标，所以是可以存储重复元素的。
* list子体系中的特有方法：

1. 增加：void add(index,element)
2. 删除：Object remove（index）
3. 查找
4. 修改：set(int index,element)返回值是当前被修改的对象

实现代码：

Public static void main(String[] args)

{

List list=new ArrayList();

methodDemo();

}

Public static void methodDemo(List list)

{

List.add("abc1");

List.add("abc2");

List.add("abc3");

List.add("abc4");

Sop(list);

//添加：

List.add(1,"abc7");//[abc1,abc7,abc2,abc3]在角标1位置往后顺延。

//删除

Object obj=list.remove(2);//删除指定索引上的元素，根据角标删除会返回来被删对象

Sop(obj);//obj：abc2 [abc1,abc7,abc3]

//获取:获取指定角标上的元素并返回，这也是和remove的区别。

Object obj=list.get(1);

Sop(obj);//obj:abc1 [abc1,abc7,abc3]

//获取元素的索引，如果元素存在返回索引，不存在返回-1，依次还可以判断元素是否存在。

Int index=list.indexOf("abc1");

Sop(index);//index=1

//获取指定的子列表sublist(fromIndex,toindex)包含头不包含尾

List=list.sublist(1,2);//[abc1]

//修改：

Object obj=list.set(1,"haha");

Sop(obj);//[abc1,haha,abc3]

记住：只有list集合支持增删改查。

}

}

listIterator:

列表迭代器：想要在迭代的过程中进行元素的增加等动作，iterator就不行了，可以使用其子接口listaIterator，而且该接口的对象只有通过list集合的listIterator方法获取，是List集合特有的取出元素方式，它包含了增删改查的方法，而iterator只有hasNext()、next()、remove（）方法

实现代码：

Public static void main(String[] args)

{

List list=new ArrayList();

List.add("abc1");

List.add("abc2");

List.add("abc3");

List.add("abc4");

ListIterator it=list.listIterator();

While(it.hasNext())  {

Object obj=it.next();

If("abc2".equals(obj))

{

It.add("abc01");//添加元素

It.set("hehe");//修改元素

}

Else

{

Sop(obj);

}

}

}

Sop(it.hasNext());//false；

与iterator的区别：

实现代码：

Iterator it=list.iterator();//迭代器是集合的功能，知道集合中有多少元素。

While(it.hasNext())

{

Object obj=it.next();

//ConcurrentModificationException

多线程导致的，不能再同时遍历的时候进行删除，原因是迭代过程中，又使用集合对象对元素进行了操作导致了并发修改异常的发生，怎么去解决？迭代器的删除方式

If("abc2".equals(obj))

{

List.remove(obj);

It.remove();用迭代器的自身的方法。

}

Else  {

Sop(obj);

}

}

* list体系结构：
  + ArrayList:低层数据结构是数组，不同步，有别于vector,替代了它，也是可延长数组，百分之五十，提升了空间的效率。查询很快，增删很慢。
  + LinkedList:低层数据结构是链表数据结构，查询速度慢，增删比较快，元素也是有角标的，以为内其父类list都有角标。
* Vector:低层数据结构是数组结构，超过长度就会创建一个新数组，长度为前数组的一倍，对其进行重新赋值。同步安全但是效率低。
  + 特有方法：

Enumeration:枚举：支持枚举接口，但是该接口因为名称过长，被Iterator取代。

实现代码：

Public stati void main(String[] arsg)

{

Vector v=new Vector();

v.addElement("abc1");

v.addElement("abc2");

v.addElement("abc3");

v.addElement("abc4");

Enumeration en=v.elements();

While(en.hasNext())  {

Sop(en.nextElement());

}

}

注意：此接口和iteration接口的功能是重复的。

ArrryList和LinkedList的分析比较：

总结：当增删操作较多时用LinkedList。当查询较多的时候用ArrayList,自有增删有查询的还是ArrayList，因为后面还是查询的使用较多。

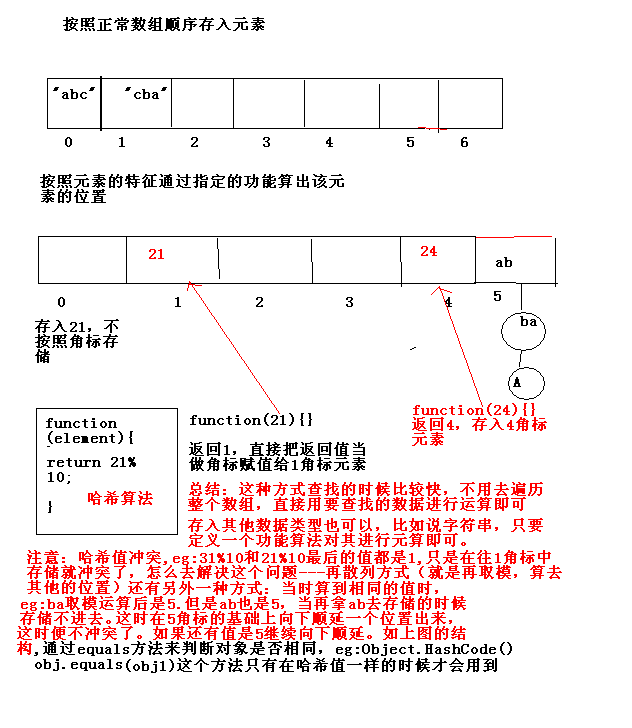
**Set集合**：无序、不重复、该接口中的方法和Collection接口中的方法一致。

HashSet：低层哈希表数据结构，不同步的，它保证元素唯一性的方式：

根据元素的两个方法来完成的，一个是hashCode、一个equals。只有当hashCode方法算出的哈希值相同时，会再次判断两个元素的equals方法是否为ture；

如果是true说明两个元素相同，不存储，所以往hashSet集合中存储自定义对象时，要覆盖hashCode、equals方法，通过自定义独享具备的特有数据来定义hashCode、equals的具体实现。

哈希结构表分析：



Hash表和数组表的比较，哈希表数据结构是按照元素的特征通过指定的功能算出该元素的位置，这种方式查找时候比较快，不 用去遍历整个数组，直接用要查找的数据进行元素即可，存入其他的数据类型也可以，比如说字符串，只要定义一个功能算法对其进行运算即可，

注意：哈希值冲突，比如31%10和21%10最后的值都是1，只是往1角标中存储就冲突了，怎么去解决这个问题呢？哈希表有种特殊的方式：再散列方式（就是再把这个数据进行取模，算出其他的位置），除此之外还有另外一种方式：当算到相同的值时，就在该冲突的位置的基础上向下顺延一个位置出来，这时候便不再冲突了，如过还有值冲突就继续向下顺延，上图结构通过equals方法来判断对象是否想用，这个方法只有在哈希值一样的时候才会用到。

实现代码：

需求：存储自定义对象，比如Person，同姓名和同年龄视为同一个人，是相同元素。

HashSet hs = new HashSet ();

hs.add(new Person("lisi1",20));

hs.add(new Person("zhangsan",20));

hs.add(new Person("wangwu",20));

hs.add(new Person("sunba",20));

hs.add(new Person("zhangsan",20));

取出来

Iterator it=hs.iterator();

While(it.hasNext())

{

Person p=(Person)it.next();

Sop(p.getName()+p.getAge());

}

Class Person()

{

Private String name;

Private int age;

Person(String name,int age)

{

This.name=name;

This.age=age;

}

Public void setName(String name){

This.name=name;

}

Public void getName()

{

Return name;

}

Public void setAge(){

This.age=age;

}

Public int getAge()  
 {

Return age;

}

Public String toString(){

Return "Person:"+name+"::"+age;

}

定义一个比较方式，按照年龄进行自然排序

Public int compareTo(Object o){

Person p=(Person)o;

Int temp=this.age-p.age;

Return temp==0?this.name.compareTo(p.name):temp;

/\*\*按照姓名进行自然排序

Int temp=this.name.compareTo(p.name){

Return temp=-0?this,.age-p.age:temp;

}

\*/

}

覆盖hashCode()方法

Public int hashCode(){

Final int NUMBER=28;//为什么要定义常量，是因为避免哈希值冲突的情况，比如说一个姓名取哈希值为20，年龄取哈希值为40.另外一个人的姓名取哈希值为40年龄哈希值为20；两个的和都是60，用它进行哈希运算，可能得到的值都是相同的，这样的话再哈希结构表中的数据存储都是冲突的。所以才定义这样一个常量类避免这种冲突。

Return name.hashCode()+age\*NUMBER;

}

覆写equals方法其实就是建立对象自身特有的判断对象是否相同的依据。

Public boolean equals(Object o){

If(!(Obj instanceof Person))

Throw new ClassCastException("数据错误"):

Person p=(Person)obj;

Return this.name.equals(p,.name)&&this.age==p.age;

}

}

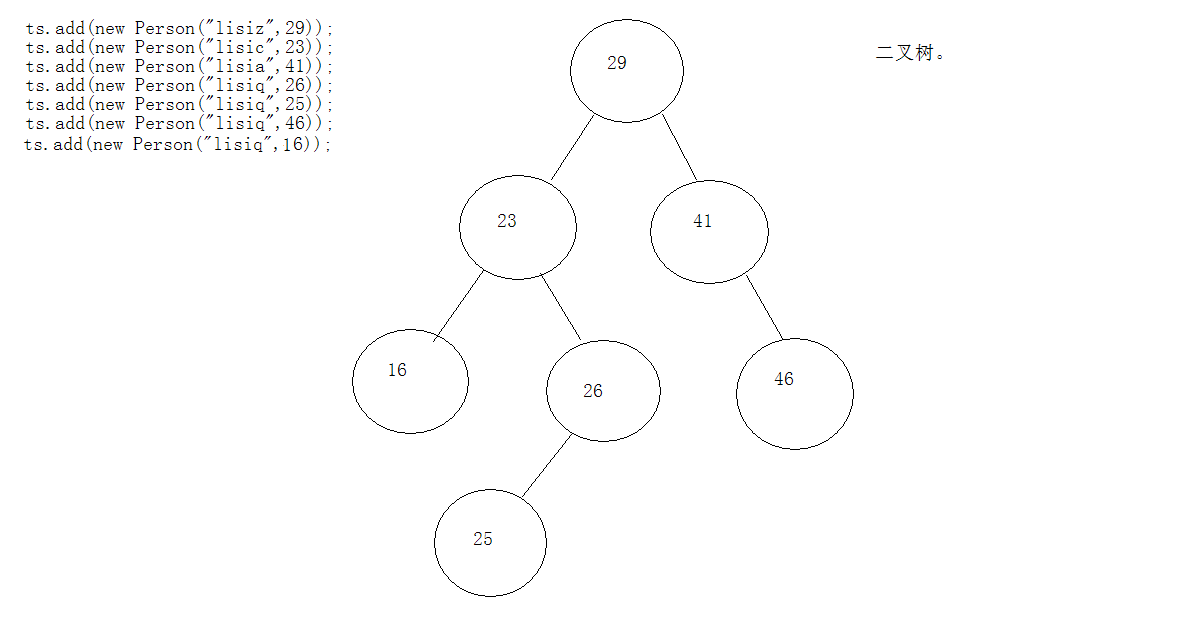
* **TreeSet：**用于给Set集合中的元素按照指定的顺序进行排序，低层是二叉树数据结构，线程是不同步的。

如何保证元素的唯一性呢？就是用过元素对象的比较方法返回值来确定的，如果为0，视为两个元素为相同的元素，不存储。

**两种排序方式：**

* + 让元素自身具备比较功能，就是强制让元素去实现comparable接口，覆盖compareTo方法。这时元素具备的自然排序。可是如果元素自身不具备比较的功能，获取具备的比较功能不是所需要的。这时该排序方式就不能用了。
  + 让集合自身具备比较功能，需要定义比较器，其实就是将实现了Comparator接口的子类对象作为参数传递给TreeSet集合的构造函数，让treeSet集合一创建就具备了比较功能。该子类必须要覆盖compare方法。

**二叉树数据结构分析**：



二叉树数据结构的特点，当第一个元素存入时，后面存入的数据都要跟它进行比较，如果小则存储在这个元素的左边的位置，如果大就存储在这个元素的右边的位置，依次下去，它在对元素进行遍历的时候，不是从上到下，而是跟你要遍历的元素与它进行比较。

**实现代码：**

TreeSet ts=new TreeSet(new ComparatorByName());

Ts.add(new Person("lisia",29));

Ts.add(new Person("lisic",30));

Ts.add(new Person("lisiz",21));

Ts.add(new Person("lisi11",41));

Iterator it=ts.iterator();

While(it.hasNext()){

Person p=(Person)it.next();

Sop(p.getName()+p.getAge());

}

**当元素自身没有比较功能的时候要定义实现Comparator接口的方法，让对象具备有比较功能,这时候要覆盖comparator接口的compare方法**

**实现代码：**

Class ComparatorByName implements Comparator{

Public int compare(Object o1,Object O2){

Person p1=（Person)o1;

Person p2=(Person)o2;

Int temp=p1.getName().compareTo(p2.getName());

Return temp==0?p1.getAge()-p2.getAge():temp;

}

}

* **查阅集合的技巧**

List

Vector

ArrayList

LinkedList

Set

HashSet  
 TreeSet

在JDk1.2集合框架出现后，名称的阅读性非常强，通过名称就可以识别。明确所属的子体系，只要看后罪名，到底具体这个集合是什么结构，有什么特点，只要看前缀即可。

看到Array:数组结构，就要想到角标，有序，可重复，查询快

看到link：链表结构，想到增删快，同时可以明确操作first last的方法、

看到hash:哈希表结构，就要想到无序，不可以重复，必须要想到元素依靠的hashCode和equals方法来保证唯一性。

看到Tree：二叉树，就要想到比较的功能和排序，必须要想到两个接口，Comparable--compareTo Comparator---compare

集合中常用的集合对象都是不同步的。

* **泛型：**

**由来：**

在jdk1.4版本之前，定义容器可以存储任意类型的对象，但是取出时，如果使用了具体元素的方法，需要向下转型的动作，因为元素类型的不一致，会导致运行时类型转换异常。

原因是：在定义集合时，并没有明确集合中的元素的具体类型，借鉴数组的定义方式就可以避免这个问题。

在JDk1.5版本以后出现了一个解决机制，忽悠事故在定义容器时，直接明确容器中的元素的类型，这既是泛型的表现，用<>格式来使用。

其实泛型就是使用到了带有<.>的类和接口时，传递需要操作的对象类型就可，其实就是传递类型参数。

* **什么时候写泛型？**

最简单的而一种体现，只要使用了带有<>的类和接口，就指定具体对象类型、

* **泛型的好处：**

1. 将运行时出现的类型转换异常问题在编译时期给解决了，运行就安全了.
2. 避免了强制转换的麻烦.

所以泛型就是JDK1,5后出现一个安全机制。

泛型擦除：泛型是编译时期的安全机制，编译时通过泛型机制，编译器多了对元素类型进行检查的步骤，如果检查通过，差生的class文件是不带有泛型的。

泛型补偿：在对元素存储的时候，可以完成类型的判断，怎么用指定的类型来接收呢？JVM在运行时，会获取元素的类型，并用该类型对元素进行转换即可。

* 泛型类：

在写容器之前想要容器中要存什么？所以要先写好泛型

Eg:TreeSet<Person> ts=new TreeSet<Person>();

有了泛型后强制转换就没有用

Eg：public class Person implements Comparator<Person>{}

对于下面覆盖的它的compare方法就不用强制转换了。

Public int compare(Person p){}这里就不用在做强制转换了。

实现代码：

Public static void main(String[] args){

Tool t=new Tool();

t.setObject(new Cat());

t.setObject(new Dog());这时编译会出现没问题，但是运行时会出现错误，因为发生了类型转换异常，dog不能转换成Cat。

Cat c=(Cat)t.getObject();

Tool<Cat> t=new Tool<Cat>();

t.setObject(new　Cat());此时就不用强转了

t.setObject(new Dog()):此时编译就失败，把运行时的异常转到编译时候。

}

Class Cat{}

Class Dog{}

/\*

Class Tool{

Private Object obj;

Public void setObject(Object obj){

This.obj=obj;

}

Public Object getObject(){

Return obj;

}

}

\*/

改进后

Class　Tool<T>{

Private T t;

Public void setObject(T t){

This.t=t;

}

Public T getObject(){

Return t;

}

}

总结：这就是将泛型定义在类上，这个类称为泛型类，是一种替代Object类的一种技术，因为Object类有自动类型提升和转换，泛型已经替代它做了，编译时期更安全，省去了强转的麻烦。

什么时候使用泛型类？当类中要操作的引用数据类型不确定的时候，以前使用的是共性类型Object现在可以使用泛型来解决，比Object更为安全。

* 泛型方法：既然类中的方法操作类型不确定，其实可以将泛型定义在方法上。

实现代码：

Tool<String> t=new Tool2<String>>();

T.show(new Integer(5));编译失败，因为类型不一致。

Tool2<Integer> t1=new Tool2<Integer>();这样便可以操作Integer类型的对象了。

T.show();

Class Tool2<T>{

显示对象的打印结构，可是对象不确定，所以使用Object，而现在可以使用泛型解决。

<W>void show(W obj){

Sop(obj);

}

当静态方法的时候，如果方法是静态的是无法访问类上定义的泛型的，如果该方法还需要泛型，必须将泛型定义在方法上，

Static <Y> void staticMethod(Y t){}

泛型定义在方法上时只能放在返回值的前面和修饰符的后面

}

什么时候使用泛型方法？当方法操作的引用数据类型不确定的时候就使用方法。

* 泛型接口：

实现代码：

Interface Inter<T>{

void show(T t);

}

Class InterImpl implements Inter<String>{

Public void show(String str){

Sop(str);

}

}

不明确的时候用上面这种，但是当我们明确了类型时候使用下面的这种情况

Class InterImpl<W> implements Inter<w>{

Public void show(W w){}

}

Public static void main(String[] args){

New InterImpl().show("hehe");

}

如果要对操作的类型进行限定，只操作一部分类型时，可以使用泛型的高级功能，泛型的限定。

？ Extends E：可以接收E类型和E的子类型。

？ SuperＥ：可以接收Ｅ类型或者Ｅ的父类型。

实现代码：

Public static void printCollection(Collection<? Extedns Person> coll){

Iterator<? extends Person> it = coll.iterator();

while(it.hasNext()){

// Student stu = it.next();

Person p = it.next();

System.out.println(p.getName()+"-----"+p.getAge());

}

}

* Map集合：

特点：

它是集合框架中的顶层。

它一次存储一对元素，一个叫键，一个叫值，也就是键值对。

Collection一次存一个，称为单列集合。

Map一次存一对，称为双列集合。

map集合中必须保证键的唯一性。

如果存储键值对时，键出现重复，新值覆盖旧值，其实map集合存储的是映射关系。

Map常见的方法：

该集合存储的键值对，一对一对的往里存，而且要保证键的唯一性。

1、添加：

put(key,value);

putAll()

2、删除：

clear();

remove(key);根据键来删，因为键唯一。

3、判断：

containKey():

containVlaue();

isEmpty();

4、获取:

get(Object key);

size();获取长度。

values();获取所有的值。张三对的北京人，李四对应上海人。所以要把所有的值都获取，看。

entrySet();将map集合中的映射关系

map.Entry其实Entry也是一个接口，它是map接口中的一个内部接口。

Interface Map{

public static interface Entry{

public abstract Object getKey();

public abstract Object getValue();

}

}

keySet();将map中所有的键都存入了set集合，因为set具备迭代器，所以可以迭代方式取出所有的键，在根据get方法获取每一个键对应的值。

Map集合的取出原理：将map集合转成set集合，在通过迭代器取出

2、Map

* Hashtable:低层是哈希表数据结构，不可以存入null作为键和null作为值。该集合是线程同步的。JDK1.0
* HashMap:低层是哈希表数据结构，该集合是不同步的，允许使用null键和null值。JDK1.2效率高
* TreeMap:低层是二叉树数据结构，线程不同步，可以用于给Map集合中的键进行排序。

面试题：collections和collection有什么区别？

1、里面的方法都是静态的。

2、没有构造函数是因为不用创建对象的。

特点：

1、sort()可以对集合类进行排序。

集合框架的工具类：

Collections:是一个类、collection是一个接口。

特点：

1、方法全部是静态的、不需要创建对象、类名直接调用。

2、能把非同步的变成同步，也就是说把不安全的变成安全的。

常用方法:

binarySearch（）：只能对list集合进行查找，为什么？因为查找的时候有角标，保证元素有顺序，返回的是：-（插入点）-1；-1是为了出现0的情况。

Max（）：求最值：

reverseOder():从大到小反序。

Fill():全部替换为自己指定的。

Shuffle():随机置换，比如书扑克牌

halfSearch():折半查找

synchronizedList():能解决线程不安全，并保证一定的高效，将非同步的集合转成同步集合。

collection和collections的区别：

collections是集合框架中的一个工具类，里面提供了操作较多的对集合进行操作的方法，比如说synchronizedList、binarySeach()等。它还可以把非同的的变成同步的，也就说把不安全的变成安全的。

Collection同样是集合框架中的一个工具类，单列集合、顶层接口、定义了单列集合的顶层方法、增删改查，有两个子接口list、set，这样两个子接口下面各有两个常用的子类：list有arrayList、linkedlist，set有hashSet、treeSet。（自己还可以扩展、越多越好）

Arrays:

* 由来：

由于数组中只有一个属性可以使用length:为了方便数组对象的使用，所以出现了arrays。

* 常见方法：

binarySearch():二分查找，重载的形式，使用该方法之前必须对数组进行排序，所以说只能对有序的数组进行使用，工具类一般是不用创建对象的，直接用类名调用就可以了。

copyOf():赋值数组中的一部分。

Equals（）：对数组对象进行判断，是否相同。

Fill():把数组中的元素全部替换成指定的值。

hashCode():算出数组的hash码。

Sort（）：排序，可以对对象进行排序。还可以对泛型进行排序。

toString（）：返回字符串。

Eg:

Int[] arr={3,4,2,8}:

Sop(Arrays.toString(arr));//[3,4,2,8]

增强特性：

asList（）：能将数组变成集合：

Eg:String[] arr={"ahas","abd","haha","xixi"};

List list=Arrays.aslist(arr);

返回一个list集合，将数组中的元素直接作为list集合的元素存在，

好处：将数组转成集合后，就可以使用集合的方法来操作数组中的元素了，因为数组只有一个属性length，而且arrays中也没有足够的多的方法去操作集合。比如说查询数组中是否包含哪个元素。

Eg:sop(list.contains("abd"));true

sop(list.indexOf("abd"));//index=1;查找它的位置。

List.add("heihei")://不支持操作异常，因为数组长度是固定的，这里尽管改造成了集合，但是还使用低层的数组，不能做改变数组长度的操作，也就说增删不能操作。

如果数组中存储的都是对象，转成集合时，会将数组中的元素作为集合中的元素存在，如果数组中的存储的都是基本数据类型转成集合时，该数组会作为集合中的元素存在。

toArray:将集合编程数组：

Eg:collection<String>coll=new Collection<String>()

Coll.add("abc");

oll.add("nba");

oll.add("xixi");

String arr=coll.toArray(new String[2]);

Sop(Array.toString(arr)):[abc.nba,xixi]

1、为什么能装四个？但是数组的长度只定义的是2个

原因：给toArray传递指定类型的数组时，长度该如何指定呢？如果长度指定值比集合中的size小，那么toArray方法内部会创建一个该类型的数组，长度和集合长度一致。用于将集合中的元素存储得到数组中，如果场地指定值比集合size大，那么toArray方法内部不会创建新数组，以传递的数组为主，超出集合的长度的部分元素默认为null.所以建议传递的数组和集合size相等。使用coll.size():

为什么要将集合变成数组？

其实是为了限定对元素的操作方法，尤其是增删。

foreach：增强for循环。

Eg：ArrayList<String>a=new ArrayList<String>():

add("abc1");

A.add("abc2");

A.add("abc3");

For(Iterator<String>it=a.iterator();it.,hasNext()){

Sop(it.,next());

}

这是最常用的迭代器方式：

增强for循环：

For(数据类型 变量名：数组或者collection集合){}

For(String str:a){

Sop(str):

}

面试题：

传统for循环和增强for循环的区别？

传统for循环可以通过增量完成循环的动作，让指定代码运行多次，增强for循环遍历必须有目标，而且目标只能是数组和collection单列集合。

增强for循环和迭代器的区别？

通常增强for循环只用于遍历目标，遍历过程中少有对元素的操作，而迭代器在遍历元素的过程中，可以对遍历的元素进行操作。比如remove。

3、注意事项：

建立遍历数组可以使用增强for循环，也可以使用传统for循环。因为遍历数组可是用增强for循环，也可以使用传统for循环。

只有遍历可以增强for循环简化书写，一旦对元素有其他操作时，建立使用for循环因为传统for里面定义角标比较方便面。不可以直接用增强for去遍历map集合，但是可以将map集合转成set集合，就可以使用增强for循环，因为set集合是collection的子集合。

代码实现：

Eg:Map<Integer,String>map=new Map<Integet,String>();

Map.put(2,"haha");

Map.put(2,"qq");

Map.put(3,"jjha");

For(Integer key:map.keySet){

String value=map.getKey();

Sop(key+value);

}

第二种方式：

For(Map.entry<Integer,String>me:entrySet()){

Integer key=me.getKey():

String value=me.getValue():

Sop(value+key);

}

可变参数：

好处：不用显示的创建数组，只要将数组中的元素作为参数进行传递即可，nebulizer会自动将这些元素封装到一个数组中，并进行传递，其实该特性的好处就是简化书写，减少了数组的定义。

弊端：

不要以为可以将所有的数组参数变成可变参数。

可变参数只能定义在参数列表的结尾。

Eg:public static void int add(int a,int b,int...nums)如果放在前面就是语法错误。

实现代码：

Eg:多个数相加：

思路：数据多了存储起来，用数组存起来

Public static void add(int[] arr){

Int sum=0;

For(int x=0;x<arr.length;x++)

Sum+=arr[x];

}

Return sum;

更为简单的方法，可变参数，其实就是一个数组。

Public static int add(int...nums)

Return 2;

* System类：

特点：

该类中的方法都是静态的，不可以被实例化。

该类可以获取JVM启动时从本地系统取到的属性信息，通过一个getProperites方法就可以获取所有的系统的信息，而且信息都是键值对的形式体现，而且键都是固定的，通过查阅这个方法的api文档说明，就可以知道键的名称，所以要获取指定的 系统平台中的信息，要通过该类来完成。

它有两个属性很重要：

一个是out属性，对应的是标准的输出设备：显示器，软件：控制台

一个in 属性：对应的标准的输入设备：键盘，所以获取键盘录入信息就要使用该属性。

获取系统当前的时间：

Long currentTimeMillis();

5、实现代码：

1、currentTimeMillis();获取当前时间。

* Runtime类：

特点：

它代表的是一个运行时实例

它并不能创建对象，因为没有构造函数

它居然有非静态的方法，说明该类中必须有静态方法，而且返回值类型都是Runtime类型

它只有一个静态方法返回了本类对象，getRuntime说明该类使用了单例。

实现代码：

问题：哪里见过单列设计？java当中本身就有单例设计的体现，Runtime对象。

eg:Runtime r=Runtime.getRuntime();

process p= r.exec("notepad");//打开记事本

p.destroy();//杀死子线程 java中只能杀它开启的， 不是它开启的杀不了。

它不仅能打开某个应用程序，还可以用这个程序打开它关联的文件

Process p=r.exec("kankan.exe E:\\java0218\\xxx.avi");

### 五、作业讲解

### 六、当日课程内容总结

### 本次直播课(后)复习巩固补充题

1.创建一个Student类 具有属性 id name age 创建ArrayList容器 要求通过方法的封装完成以下操作，

* 1. 循环录入5条数据
  2. 在集合首位添加一条数据
  3. 删除集合末尾的数据并且打印出删除学生的信息
  4. 通过名字查询出符合条件的学员姓名
  5. 输入id删除一条学员信息

2.在Arraylist中存放以下数据： “张三”、“李四”、“王五”、“张三”、“aaa”、“aaa”、“bbb”

要求：去除ArrayList中的重复元素。

3.给定一系列字符串： "chenhao" "zhangsan" "zhangsan" "chenhao" "lisi" "wangwu" "zhaoliu" "xiaoqiang" "haha"

打印输出各个字符串出现的次数（使用HashMap实现，键保存字符串，值保存保存出现的次数）

### 八、下次自主学习任务布置



需要通过平台提交作业。完成平台上所布置的作业。