集合框架

集合框架（数据结构和多线程）

集合框架（数据结构和多线程）

java.util.*；包下（工具包）

考题集中！！（源码理解，整个框架认识）

概念：动态数组 --解决数组长度固定。

-动态数组：当元素个数达到最大值时，动态增加容量。

List 接口：

1.ArrayList 与Vector:区别

（源码理解，多线程同步理解，动态扩容机制，懒加载等）

2.ArrayList 线程不安全的List集合 是否了解JUC包下的线程安全List(CopyOnWriteArrayList)

set：
1.set集合与map集合的关系
2.hashCode.equals方法关系
3.Comparable,Compartor的关系

Map:
1.请对比HashMap,Hashtable关系
2.是否了解ConcurrentHashMap以及实现。

1.类集的产生-JDK1.2

1.1概念

类集是一个动态数组，解决数组定长问题。
数据结构：java版

2.Collection接口-单个对象保存的最顶层父接口

特点：collection接口在每次进行数据处理操作时只能够对单个对象进行处理
里面包含泛型：为了类集服务的（解决向下转型问题的）JDK1.5

```
public interface Collection<E> extends Iterable<E> {
```

```
Iterable<E>：迭代器就是为了遍历集合
```

```
Iterator<T> iterator();（取得集合的迭代器）
JDK1.5之前直接写在Collection中
```

2.1Collection中提供的一些核心方法

No.	方法名称	类型	描述
1.	public boolean add(E e);	普通	向集合中添加数据
2.	public boolean addAll(Collection<? extends E> c);	普通	向集合中添加一组数据
3.	public void clear();	普通	清空集合数据
4.	public boolean contains(Object o);	普通	查找数据是否存在，需要使用 equals()方法
5.	public boolean remove(Object o);	普通	删除数据，需要 equals()方法
6.	public int size();	普通	取得集合长度
7.	public Object[] toArray();	普通	将集合变为对象数组返回
8.	public Iterator<E> iterator();	普通	取得 Iterator 接口对象，用于集合输出

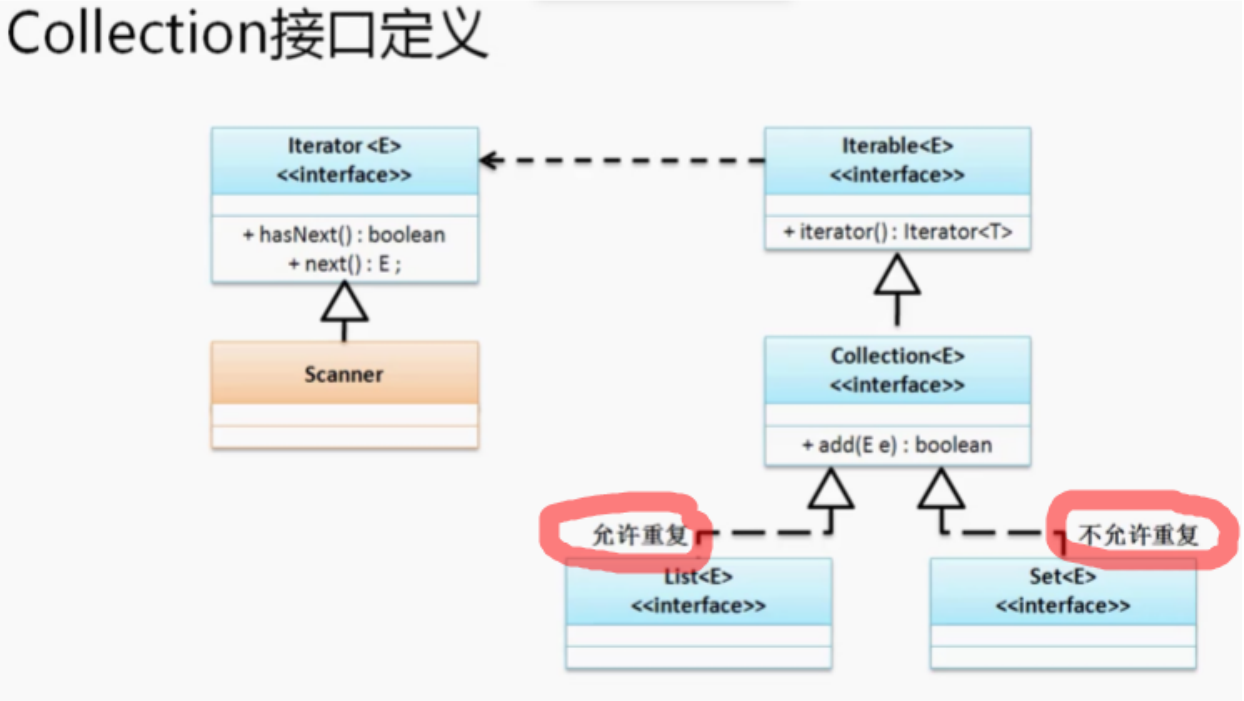
```
1.add(T t):方法向类集中添加元素
```

```
2.Iterator<T> iterator();（取得集合的迭代器）
```

Collection接口之**定义了一个存储数据的标准，但无法区分存储类型。**

因此在实际中我们往往使用两个子接口 **List（允许保存重复数据）**、**Set(不允许数据重复)**。一般不直接使用Collection接口。

2.2Collection的接口定义：



2.3List接口（80%）-允许数据重复

在进行单个集合处理，优先考虑List接口

在List接口中拓展了**两个重要的，独有的方法(List接口独有)**

```
1 public E get(int index):根据索引下标取得数据
```

```
1 public E set（int index)：根据索引下标更新数据，返回修改之前的数据
```

List接口有**三个重要的子类 ArrayList（90%） ,Vector,LinkedList**

List接口**要想保存自定义类的对象**，该类**必须覆写equals方法**来使用我们 **contain()、remove()** 等方法。

2.4面试题一

***********ArrayList,Vector,LinkedList的区别*************

2.4.1ArrayList,Vector区别： （5点法）

1.出现版本：

ArrayList:JDk1.2

Vector：JDk1.0(出现在List,Collection接口之前)

2.调用无参构造的区别(初始化策略)

Vector在无参构造执行后**将对象数组大小初始化为10**

ArrayList采用**懒加载策略**，在构造方法阶段并不初始化对象数组。

在**第一次添加元素时才初始化对象数组大小为10（初始化策略）**

- [ ] **源码刨析这句话**

源码刨析

Vector（初始化策略）：

```
1  public Vector(int initialCapacity, int capacityIncrement) {
2   super();
3   if (initialCapacity < 0)
4   throw new IllegalArgumentException("Illegal Capacity: "+
5   initialCapacity);
6   this.elementData = new Object[initialCapacity]; //初始化数组
7   this.capacityIncrement = capacityIncrement; //增量
8  }
9
10  /**
11   * Constructs an empty vector with the specified initial capaci
 ty and
12   * with its capacity increment equal to zero.
13   *
14   * @param initialCapacity the initial capacity of the vector
15   * @throws IllegalArgumentException if the specified initial ca
 pacity
16   * is negative
17   */
18  public Vector(int initialCapacity) {
19   this(initialCapacity, 0); //initialCapacity=10 Vector容量
20  }
21
```

```
22 /**
23  * Constructs an empty vector so that its internal data array
24  * has size {@code 10} and its standard capacity increment is
25  * zero.
26  */
27 public Vector() {
28  this(10); //调用有参构造
29 }
```

ArrayList(初始化策略)：

```
1 private static final Object[] DEFAULTCAPACITY_EMPTY_ELEMENTDATA
 = {};
```

```
1 public ArrayList() { //默认无参构造时懒加载（用时增加）
2  this.elementData = DEFAULTCAPACITY_EMPTY_ELEMENTDATA;
3 }
```

3.扩容策略

ArrayList 扩容时，新数组大小变为原来数组的1.5倍。

> **假如要设置的值超出1.5倍旧值时 将容量设置为需要的值**

ArrayList 源码详细分析：

ArrayList及扩容策略..析.txt
2.54KB

Vector:扩容时，新数组大小变为原来数组的2倍。

> **假如要设置的值超出1.5倍旧值时 将容量设置为需要的值**
> **capacityIncrement:增容策略**
> **如不设置增容策略则是增加两倍**

ArrayList 源码详细分析：

Vector扩容策略源代..析.txt
2.62KB

对比

ArrayList:

```
int newCapacity = oldCapacity + (oldCapacity >> 1);1.5倍
假如要设置的值超出1.5倍旧值时 将容量设置为需要的值
```

Vector

```
int newCapacity = oldCapacity + ((capacityIncrement > 0) ?
 capacityIncrement : oldCapacity);
 capacityIncrement:增容策略
 如不设置增容策略则是增加两倍
```

4.线程安全问题：

ArrayList采用异步处理，线程不安全，效率较高；

```
public boolean add(E e) {
 ensureCapacityInternal(size + 1); // Increments modCount!!
 elementData[size++] = e;
 return true;
}
```

Vector采用在方法上加锁，线程安全，效率较低；

```
public synchronized boolean add(E e) {
 modCount++;
 ensureCapacityHelper(elementCount + 1);
 elementData[elementCount++] = e;
 return true;
}
```

(即便要使用线程安全的List,也不用Vector)?

5.在遍历上的区别：

Vector可以使用较老的迭代器的Enumration,ArrayList不支持

输出形式：ArrayList支持Iterator、ListIterator、foreach；Vector支持Iterator、ListIterator、 foreach、Enumeration

- [ ] 了解Enumeration遍历方式

2.4.2ArrayList,Vector共同点：

1.ArrayList,Vector的都实现了List接口

2..底层都使用数组实现

2.4.3ArrayList、LinkedList区别：

LinkedList底层采用双向链表实现，ArrayList底层是采用数组实现。

3.Set接口

set接口：不允许数据重复（set接口就是value值相同得Map集合）

set接口：没有扩充方法

3.1set接口常用子类

3.1.1HashSet:（无序存储）-本质上HashMap

- [ ] 无序在哪里（Map ）

按照桶的顺序进行打印（哈希策略）

> 1.底层使用哈希表+红黑树
> 2.允许存放null,无须存储

3.1.2TreeSet： （有序存储）Comparable Compartor - 本质上TreeMap

判断元素是否重复，所以Treeset 或者Treemap要使用（add()）则必须先要调用比较方法。

因为它不允许有重复的数据。（所以添加时需要，进行比较操作，所以自定义类要想保存到TreeSet中要么实现comparable。

> 要么向TreeSet传入比较器（Compartor接口））

> 1.底层使用红黑树
> 2.不允许出现空箱，有序存储
> 3.自定义类要想保存到TreeSet中要么实现comparable,
> 要么向TreeSet传入比较器（Compartor接口）

Comparable 接口和 Compartor接口区别：
在java中，要想实现自定义类的比较，提供以下两个接口

3.1.3 java.lang.Comparable接口（内部比较器）--排序接口：

1.若一个类实现comparable接口就意味着**该类支持排序。**
并且**存放该类的(Collection)**或者**对象数组**，可以直接通过**Collection.sort()或Array.sort进行排序。**

2.实现了Comparable接口的类可以直接存放在 TreeSet或 TreeMap中。

```
1 public int compareTo(T o);
```

3种返回值：
返回正数：当前对象大于目标对象

返回0：当前对象等于目标对象

返回负数：当前对象小于目标对象

3.1.4Comparator(外部排序接口)

若要控制某个自定义类的顺序，而该类本身不支持排序（类本身没有实现Comparable接口）。我们可以建立一个该类的“比较器”来进行排序。比较器实现Comparator接口即可。
“比较器”:实现了Comparator接口的类作为比较器，通过该比较器来进行类的排序。

```
1 int compare(T o1, T o2);
```

返回值与compareTo返回值完全一样

返回正数：o1>o2
返回0： o1=o2
返回负数：o1<o2

总结：

> 实现了Comparable接口进行第三方排序-策略模式，此方法更加灵活。可以轻松改变策略进行第三方的排序算法。

3.1.4 Comparable接口与Comparator接口的关系：

1.Comparable是一个排序接口，若一个类实现了Comparator接口，意味着该类支持排序，是一个内部类比较器（自己去和别人比）

2.Comparator接口是比较器接口，自定义的类的本身不支持排序，需要有若干个第三方比较器（实现了Compartor接口的类）进行类的排序，是一个外部比较器（策略模式）。

3.2重复元的比较

3.2.1重复元素的比较（TreeSet）

TreeSet与TreeMap依靠Compartor或Comparable接口来区分重复元素。

自定义类要想保存在TreeSet或TreeMap中：
I.要么该类直接实现Comparable接口，覆写Compareto方法
II.要么实现一个比较器传入TreeSet或TreeMap来进行外部比较。

3.2.2重复元素比较（HashSet）

而 HashSet与HashMap**并不依赖比较接口**，此时要想**区分自定义元素是否重复**，**需要同时覆写equals与hashCode方法。**

首先要覆写equals（）方法来判定两个元素内容是否相等。

覆写equals方法原则：
1.自反性：对于任何非空对象引用值x,x.equals(x)都返回true
2.对称性：对于任何非空的x,y,当且仅当x.equals(y)返回true,y.equals(x)也返回true;
3.传递性：对于任何非空的x,y,z，如果x.equals(y)返回true,y.equals(z)返回true,一定有x.equals（z）返回true

4.一致性：对于任何非空的x,y,若x与y中属性没有改变，则多次调用x.equals（y）始终返回true或false。
5.非空性：对于任何非空引用x,x.equals(null)一定返回false

> 先**调用hashCode**计算对象hash码决定存放的数据桶（**保证两个相同数据放在同一个桶里**）
> 而后**使用equals来比较元素是否相等。**
> 若相等，则不再放置元素；
> 若equals返回false,则再相同桶之后，使用链表将若干元素链起来（**拉链法**）。
>
> Object类提供的hashCode方法默认使用**对象的地址进行hash。**

结合源码总结

> 若两个对象**equals方法返回true他们得到hashcode必然要保证相同**，但是两**个对象的hashCode相等，equals不一定相等。**
> 当且**仅当equals与hashCode方法均返回true**，才认为这两个**对象真正相等。**

哈希表：
为何要分桶来存放元素。

4.集合输出（迭代器输出）-Iterator接口

迭代器为了遍历集合而生

Iterator接口两个核心方法：

```
boolean hasNext(); /判断是否还有下一个元素
```

```
E next(); //取得下一个元素
```

集合输出一共有以下形式：

4.1迭代输出Iterator -只能从前向后也可以从后向前（Collection接口提供）

调用Collection集合子类的Iterator方法取得内置的迭代器，使用以下输出格式

```
while(iterator.hasNext()){
 System.out.println(iterator.next());
}
```

4.2双向迭代接口ListIterator-List接口提供，set不支持

除了hasNext与next方法外还有如下方法

```
hashPrevious():判断是否有上一个元素
```

```
pervious:取得上一个元素
```

```
while(iterator.hasPreviour()){
 System.out.println(iterator.pervious);
}
```

要想使用从后向前遍历，首先至少要从前向后遍历一遍才能使用从后向前遍历。

4.3Enumeration枚举输出-Vector类支持

Vector的elements（）方法取得Enumeration对象

```
hasMoreElements()：判断是否有下一个元素
```

```
nextElements():取得下一个元素
```

4.4for - each输出（所有子类都满足）

能使用foreach输出的本质在于各个集合都内置了迭代器。

5.fail-fast机制（快速失败机制）

ConcureentModificationException发生在Collection集合使用迭代器遍历时，**使用了集合类提供的修改内容的方法报错**，而如果使用**iterator的迭代器的remove()不会出现此错误。**

- [ ] 修改iterator,会反应在原本的集合当中吗?
- [ ] 为什么?

解答：产生错误的原因

```
final void checkForComodification() {
 if (modCount != expectedModCount)
```

```
 throw new ConcurrentModificationException();
 }
}
```

Collection集合中的modCount表示当前集合修改的次数（集合的一个属性）

expectedModCount 是迭代器中记录当前集合的修改次数(迭代器的一个属性)

当取得集合迭代器时 list.iterator()，int expectedModCount = modCount;（才赋值）（调用构造方法时才会赋值）。换言之，迭代器就是当前集合的一个副本。

```
int expectedModCount = modCount;//迭代器中代码
```

快速失败策略保证了所有用户在进行迭代遍历时，拿到的数据一定是最新的数据（避免”脏读“产生）。

6.fail-safe

：不产生ConcurrentModeifiicationException异常
juc包下所有线程安全集合（CopyOnWriteArrayList）

总结：以后在迭代器遍历时，不要修改集合内容。

5.Map集合（使用版）

Map接口是java中保存**二元偶对象（键值对）**的最顶层接口

```
public interface Map<K,V> {
```

特点：key值唯一，通过一个key值一定能唯一找到一个value值。

5.1Map中的核心方法

```
public V put (K key,V value):向Map集合中添加数据
public V get(K key):根据指定的key值取得相应的value值，若没有此key值，返回null

public Set <Map.Entry<K,V>> entrySet():将Map集合变为Set集合.
```

```
5
6 public Set<K> keySet :返回所有key值得集合，key不重复。
7 public collection<V> values():  反回所有value值，value可以重复。
```

Map接口中有如下常用四个子类
HashMap（使用频率最高的-必考）,TreeMap,Hashtable,ConcurrentHashMap

5.2 Map接口得使用

//当key值重复时，再次put变为相应得value更新得操作。
//key值不存在则反回null

5.3 HashMap-类比HashSet（面试题）

HashMap
1.允许key和value为null,且key值有且只有一个为null，value允许有任意多个为null。
2.版本号 JDK1.2
3.异步处理，效率高，线程不安全
4.底层哈希表+红黑树（JDK1.8）

Hashtable(古老类)
1.key与value均不能为null
2.版本号：JDK1.0
3.使用方法加锁，效率i低，线程安全
4.底层哈希表