# 基础类

## Object

Object中包含的方法:

1.protected Object clone() //创建并返回此对象的一个副本。

2.boolean equals(Object obj) //指示某个其他对象是否与此对象“相等”。

实现其实就是用 == 判断的其实是地址。比较的是栈中的值。（原型数据相同或相同引用）

3.protected void finalize() //当垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收器调用此方法。

finalize会在对象被垃圾回收时由垃圾回收器调用，垃圾对象是指没有引用指向的对象。

4.Class<? extends Object> getClass() //返回一个对象的运行时类。

5.int hashCode() //返回该对象的哈希码值。

6.void notify() //唤醒在此对象监视器上等待的单个线程。

7.void notifyAll() //唤醒在此对象监视器上等待的所有线程。

8.String toString() //返回该对象的字符串：类名 @ 内存地址。

getClass().getName() + "@" + Integer.toHexString(hashCode())

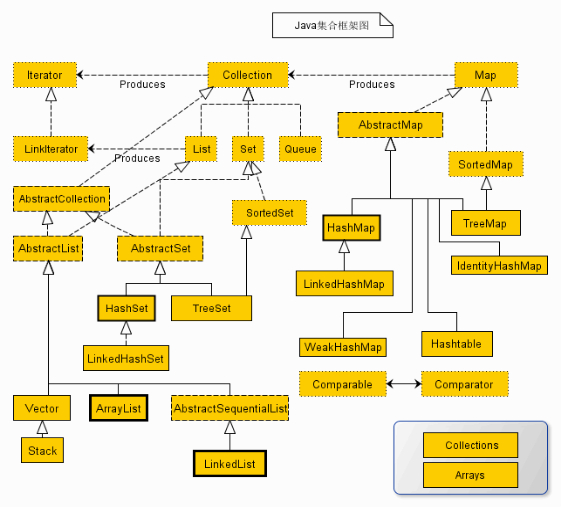
9.void wait() //导致当前的线程等待，直到其他线程调用此对象的 notify() 方法或 notifyAll() 方法。

void wait(long timeout) //导致当前的线程等待，直到其他线程调用此对象的 notify() 方法或notifyAll() 方法，或者超过指定的时间量。

void wait(long timeout, int nanos) //导致当前的线程等待，直到其他线程调用此对象的notify() 方法或 notifyAll() 方法，或者其他某个线程中断当前线程，或者已超过某个实际时间量。

## String

# 集合类



Collection接口是集合类的根接口，Java中没有提供这个接口的直接的实现类。但是却让其被继承产生了两个接口，就是Set和List。Set中不能包含重复的元素。List是一个有序的集合，可以包含重复的元素，提供了按索引访问的方式。

Map是Java.util包中的另一个接口，它和Collection接口没有关系，是相互独立的，但是都属于集合类的一部分。Map包含了key-value对。Map不能包含重复的key，但是可以包含相同的value。

Iterator，所有的集合类，都实现了Iterator接口，这是一个用于遍历集合中元素的接口，主要包含以下三种方法：

1.hasNext()是否还有下一个元素。

2.next()返回下一个元素。

3.remove()删除当前元素。

## 1、List（有序、可重复）

List里存放的对象是有序的，同时也是可以重复的，List关注的是索引，拥有一系列和索引相关的方法，查询速度快。因为往list集合里插入或删除数据时，会伴随着后面数据的移动，所有插入删除数据速度慢。

### ArrayList

基于动态数组的数据结构，用get和set方法是花费常数时间的，但是如果插入元素和删除元素，除非插入和删除的位置都在表末尾，否则代码开销会很大，因为里面需要数组的移动。

ArrayList 初始化大小是 10，扩容点规则是，新增的时候发现容量不够用了，就去扩容，扩容大小规则是，扩容后的大小= 原始大小+原始大小/2 + 1。

### LinkedList

基于双向链表的数据结构，get会非常消耗资源，除非位置离头部很近。但是插入和删除元素花费常数时间。

没有初始化大小，也没有扩容的机制，就是一直在前面或者后面新增就好。

## 2、Set（无序、不能重复）

Set里存放的对象是无序，不能重复的，集合中的对象不按特定的方式排序，只是简单地把对象加入集合中。

### HashSet

其底层其实是包装了HashMap去实现。Set集合是不允许重复元素的，HashSet需要同时通过equals和HashCode来判断两个元素是否相等，具体规则是，如果两个元素通过equals为true，并且两个元素的hashCode相等，则这两个元素相等（即重复）。所以如果要重写保存在HashSet中的对象的equals方法，也要重写hashCode方法。

HashSet hs = new HashSet();

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

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

### TreeSet

TreeSet实现了SortedSet接口，顾名思义这是一种排序的Set集合，本质上是一个红黑树原理。红黑树算法的规则: 左小右大。正因为它是排序了的，所以相对HashSet来说，TreeSet提供了一些额外的按排序位置访问元素的方法，例如first(), last(), lower(), higher(), subSet(), headSet(), tailSet()。

TreeSet集合排序的两种方式：1、让元素自身具备比较性，需要元素实现Comparable接口，重写compareTo方法，也就是让元素自身具备比较性，这种方式叫做元素的自然排序也叫做默认排序。2、让容器自身具备比较性，定义一个类实现Comparator 接口，覆盖compare方法，当Comparable比较方式，及Comparator比较方式同时存在，以Comparator比较方式为主。通过compareTo或者compare方法中的来保证元素的唯一性。添加的元素必须要实现Comparable或Comparator接口。当compareTo()或compare()函数返回值为0时，说明两个对象相等，此时该对象不会添加进来。

TreeSet ts = new TreeSet();

ts.add("ccc");

ts.add("aaa");

### LinkedHashSet

LinkedHashSet是HashSet的一个子类，LinkedHashSet也根据HashCode的值来决定元素的存储位置，但同时它还用一个链表来维护元素的插入顺序，插入的时候即要计算hashCode又要维护链表，而遍历的时候只需要按链表来访问元素。

LinkedHashSet本质上也是从LinkedHashMap而来，LinkedHashSet的所有方法都继承自HashSet, 而它能维持元素的插入顺序的性质则继承自LinkedHashMap。

## 3、Map（键值对、键唯一、值不唯一）

Map集合中存储的是键值对，键不能重复，值可以重复。根据键得到值，对map集合遍历时先得到键的set集合，对set集合进行遍历，得到相应的值。

### HashMap

Iterator，所有的集合类，都实现了Iterator接口，这是一个用于遍历集合中元素的接口，主要包含以下三种方法：

1.hasNext()是否还有下一个元素。

2.next()返回下一个元素。

3.remove()删除当前元素。

# IO类

Java平台允许我们在内存中创建可复用的Java对象，但一般情况下，只有当JVM处于运行时，这些对象才可能存在，即，这些对象的生命周期不会比JVM的生命周期更长。但在现实应用中，就可能要求在JVM停止运行之后能够保存(持久化)指定的对象，并在将来重新读取被保存的对象。Java对象序列化就能够帮助我们实现该功能。

使用Java对象序列化，在保存对象时，会把其状态保存为一组字节，在未来，再将这些字节组装成对象。必须注意地是，对象序列化保存的是对象的”状态”，即它的成员变量。由此可知，对象序列化不会关注类中的静态变量。

### Serializable

1、在Java中，只要一个类实现了java.io.Serializable接口，那么它就可以被序列化。

2、通过ObjectOutputStream和ObjectInputStream对对象进行序列化及反序列化

3、虚拟机是否允许反序列化，不仅取决于类路径和功能代码是否一致，一个非常重要的一点是两个类的序列化 ID 是否一致（就是 private static final long serialVersionUID）

4、序列化并不保存静态变量。

5、要想将父类对象也序列化，就需要让父类也实现Serializable 接口。

6、Transient 关键字的作用是控制变量的序列化，在变量声明前加上该关键字，可以阻止该变量被序列化到文件中，在被反序列化后，transient 变量的值被设为初始值，如 int 型的是 0，对象型的是 null。

7、服务器端给客户端发送序列化对象数据，对象中有一些数据是敏感的，比如密码字符串等，希望对该密码字段在序列化时，进行加密，而客户端如果拥有解密的密钥，只有在客户端进行反序列化时，才可以对密码进行读取，这样可以一定程度保证序列化对象的数据安全。

### ObjectOutputStream、ObjectInputStream

1、ByteArrayOutputStream序列化，ByteArrayInputStream反序列化：

ObjectOutputStream oos = new ObjectOutputStream(new ByteArrayOutputStream());

oos.writeObject(object);

byte[] resultByte = bos.toByteArray();

ObjectInputStream ois = new ObjectInputStream (new ByteArrayInputStream ());

ois.writeObject(object);

Object resultObject = ois.readObject();

2、FileOutputStream序列化，FileInputStream反序列化：

//Write Obj to File

ObjectOutputStream oos = null;

try {

oos = new ObjectOutputStream(new FileOutputStream("tempFile"));

oos.writeObject(user);

} catch (IOException e) {

e.printStackTrace();

} finally {

IOUtils.closeQuietly(oos);

}

//Read Obj from File

File file = new File("tempFile");

ObjectInputStream ois = null;

try {

ois = new ObjectInputStream(new FileInputStream(file));

User newUser = (User) ois.readObject();

System.out.println(newUser);

} catch (IOException e) {

e.printStackTrace();

} catch (ClassNotFoundException e) {

e.printStackTrace();

} finally {

IOUtils.closeQuietly(ois);

try {

FileUtils.forceDelete(file);

} catch (IOException e) {

e.printStackTrace();

}

}