

## 泛型

### ● 泛型的好处

- 1) 编译时, 检查添加元素的类型, 提高了安全性
- 2) 减少了类型转换的次数, 提高效率 [说明]

✓ 不使用泛型

Dog -加入-> Object -取出-> Dog //放入到ArrayList 会先转成 Object, 在取出时, 还需要转换成Dog

✓ 使用泛型

Dog -> Dog -> Dog // 放入时, 和取出时, 不需要类型转换, 提高效率

### 3) 不再提示编译警告

```
int a = 10;
```

老韩理解: 泛型 => Integer, String, Dog

- 1) 泛型又称参数化类型, 是Jdk5.0 出现的新特性, 解决数据类型的安全性问题
- 2) 在类声明或实例化时只要指定好需要的具体的类型即可。
- 3) Java泛型可以保证如果程序在编译时没有发出警告, 运行时就不会产生 ClassCastException异常。同时, 代码更加简洁、健壮
- 4) 泛型的作用是: 可以在类声明时通过一个标识表示类中某个属性的类型, 或者是某个方法的返回值的类型, 或者是参数类型。[有点难, 举例 [Generic03.java](#)]

## 泛型的语法

### ● 泛型的声明

interface 接口<T>{} 和 class 类<K,V>{}  
//比如: List, ArrayList

说明:

- 1) 其中, T,K,V不代表值, 而是表示类型。
- 2) 任意字母都可以。常用T表示, 是Type的缩写

### ● 泛型的实例化:

要在类名后面指定类型参数的值 (类型)。如:

- 1) List<String> strList = new ArrayList<String>(); [举例说明]
- 2) Iterator<Customer> iterator = customers.iterator();

### ● 泛型使用的注意事项和细节 [GenericDetail.java](#)

1. interface List<T>{} , public class HashSet<E>{}.. 等等

说明: T, E 只能是引用类型

看看下面语句是否正确?:

```
List<Integer> list = new ArrayList<Integer>();
```

```
List<int> list2 = new ArrayList<int>();
```

2. 在指定泛型具体类型后, 可以传入该类型或者其子类类型

3. 泛型使用形式

```
List<Integer> list1 = new ArrayList<Integer>();
```

```
List<Integer> list2 = new ArrayList<>(); [说明:]
```

3. 如果我们这样写 List list3 = new ArrayList(); 默认给它的 泛型是 [<E> E就是 Object ]  
即:

## 自定义泛型

## ● 自定义泛型类 (难度)

### ➢ 基本语法

```
class 类名<T, R...> {  
    成员  
}
```

### ➢ 注意细节

- 1) 普通成员可以使用泛型 (属性、方法)
- 2) 使用泛型的数组, 不能初始化
- 3) 静态方法中不能使用类的泛型
- 4) 泛型类的类型, 是在创建对象时确定的(因为创建对象时, 需要指定确定类型)
- 5) 如果在创建对象时, 没有指定类型, 默认为Object

//老韩解读

- //1. Tiger 后面泛型, 所以我们把 Tiger 就称为自定义泛型类
- //2. T, R, M 泛型的标识符, 一般是单个大写字母
- //3. 泛型标识符可以有多个.
- //4. 普通成员可以使用泛型 (属性、方法)
- //5. 使用泛型的数组, 不能初始化
- //6. 静态方法中不能使用类的泛型

```
class Tiger<T, R, M> {
```

## ● 自定义泛型方法

### ➢ 基本语法

```
修饰符 <T,R...> 返回类型 方法名(参数列表) {  
}
```

### ➢ 注意细节

1. 泛型方法, 可以定义在普通类中, 也可以定义在泛型类中
2. 当泛型方法被调用时, 类型会确定
3. `public void eat(E e) {}`, 修饰符后没有 `<T,R...>` eat 方法不是泛型方法, 而是使用了泛型

➢ 应用案例 CustomMethodGeneric.java  
看老师演示.

```
class Bird<T,R,M>{  
    public<E> void fly(E t){  
        System.out.println(t);  
    }  
}  
  
class Fish {  
    public<A> A show(A t){  
        System.out.println("t的值: "+t);  
        System.out.println("t的类型: "+t.getClass().getSimpleName());  
        return t;  
    }  
}
```

韩顺平  
教育

## 复习枚举

### 1. 自定义类实现枚举

1) 将构造器私有化; 2) 本类内部创建一组对象; 3) 对外暴露对象 (通过为对象添加 `public static final` 修饰符); 4) 可以提供 `get` 方法, 但不要提供 `set` 方法。

```
package com.wengan.enum_;
```

```
/**
```

```
 * @author 文湓
```

```

* @version 1.0
*/
public class Enum01 {
    public static void main(String[] args) {

        System.out.println(Season.SPRING);
        System.out.println(Season.SUMMER);
        System.out.println(Season.AUTUMN);
        System.out.println(Season.WINTER);

    }
}

class Season{
    private String name;
    private String desc;//描述

    private Season(String name, String desc) {
        this.name = name;
        this.desc = desc;
    }

    //定义四个对象
    public static final Season SPRING = new Season("春天","温暖");
    public static final Season SUMMER = new Season("夏天","炎热");
    public static final Season AUTUMN = new Season("秋天","凉爽");
    public static final Season WINTER = new Season("冬天","寒冷");

    public String getName() {
        return name;
    }

    public String getDesc() {
        return desc;
    }

    @Override
    public String toString() {
        return name + desc;
    }
}

```

用 enum 关键字:

1) 使用关键字 enum 替代 class

- 2) `public static final Season SPRING = new Season("春天", "温暖");` 直接使用 `SPRING("春天", "温暖")`
- 1) 如果有多个常量（对象），使用逗号间隔即可
- 2) 如果使用 `enum` 来实现枚举，要求将定义常量对象放在最前面

### enum常用方法一览表

方法名	详细描述
<code>valueOf</code>	传递枚举类型的 Class 对象和枚举常量名称给静态方法 <code>valueOf</code> ，会得到与参数匹配的枚举常量。
<code>toString</code>	得到当前枚举常量的名称。你可以通过重写这个方法来使得到的结果更易读。
<code>equals</code>	在枚举类型中可以直接使用 <code>"=="</code> 来比较两个枚举常量是否相等。Enum 提供的这个 <code>equals()</code> 方法，也是直接使用 <code>"=="</code> 实现的。它的存在是为了在 Set、List 和 Map 中使用。注意， <code>equals()</code> 是不可变的。
<code>hashCode</code>	Enum 实现了 <code>hashCode()</code> 来和 <code>equals()</code> 保持一致。它也是不可变的。
<code>getDeclaringClass</code>	得到枚举常量所属枚举类型的 Class 对象。可以用它来判断两个枚举常量是否属于同一个枚举类型。
<code>name</code>	得到当前枚举常量的名称。建议优先使用 <code>toString()</code> 。
<code>ordinal</code>	得到当前枚举常量的次序。
<code>compareTo</code>	枚举类型实现了 <code>Comparable</code> 接口，这样可以比较两个枚举常量的大小（按照声明的顺序排列）。
<code>clone</code>	枚举类型不能被 Clone。为了防止子类实现克隆方法，Enum 实现了一个仅抛出 <code>CloneNotSupportedException</code> 异常的不变 <code>Clone()</code> 。

1. `toString`: Enum 类已经重写过了，返回的是当前对象名，子类可以重写该方法，用于返回对象的属性信息
2. `name`: 返回当前对象名（常量名），子类中不能重写
3. `ordinal`: 返回当前对象的位置号，默认从 0 开始
4. `values`: 返回当前枚举类中所有的常量
5. `valueOf`: 将字符串转换成枚举对象，要求字符串必须为已有的常量名，否则报异常！
6. `compareTo`: 比较两个枚举常量，比较的就是位置号！

```
Season2 autumn = Season2.AUTUMN;
System.out.println(autumn);
System.out.println(autumn.name());
System.out.println(autumn.ordinal()); //2
Season2[] values = Season2.values();
for (Season2 season2 : values) { //增强for
    System.out.println(season2);
}
Season2 value = Season2.valueOf("SPRING");
System.out.println(value);
System.out.println(autumn.compareTo(value));
```

```
package com.wengan.enum_;

/**
 * @author 文潞
 * @version 1.0
 */
public class EnumMethod {
    public static void main(String[] args) {
        Season2 autumn = Season2.AUTUMN;
        System.out.println(autumn.name());
        System.out.println(autumn.ordinal()); //输出该常量对象的次序(编号)
        //values 返回的是 Season2[], 含有所有的枚举对象
        Season2[] values = Season2.values();
        //增强 for 循环, 依次从 values 数组中取出数赋给 season2
        //如果取值完毕就退出 for 循环
        for (Season2 season2: values) {
```



```

        System.out.println(season2);
    }

    //valueOf 将字符串转换成枚举对象，要求字符串必须为已有的常量名，否则报告异常
    Season2 autumn2 = Season2.valueOf("AUTUMN");
    System.out.println("autumn" + autumn2);

    //compareTo 比较两个枚举常量，比较的是编号
    //Season2.AUTUMN 的编号 - Season2.SUMMER 的编号
    System.out.println(Season2.AUTUMN.compareTo(Season2.SUMMER));
}
}

```

**老韩小结:** 底层是 `StringBuilder sb = new StringBuilder(); sb.append(a); sb.append(b);` `sb`是在堆中，并且`append`是在原来字符串的基础上追加的。  
**重要规则**，`String c1 = "a" + "b" + "cd"`；常量相加，看的是池。`String c1 = a + b`；变量相加，是在堆中

- `equals` // 区分大小写，判断内容是否相等
- `equalsIgnoreCase` // 忽略大小写的判断内容是否相等
- `length` // 获取字符的个数，字符串的长度
- `indexOf` // 获取字符在字符串中第1次出现的索引，索引从0开始，如果找不到，返回-1
- `lastIndexOf` // 获取字符在字符串中最后1次出现的索引，索引从0开始，如找不到，返回-1
- `substring` // 截取指定范围的子串
- `trim` // 去前后空格
- `charAt`: 获取某索引处的字符，注意不能使用`Str[index]` 这种方式。

```

String str = "hello" ;
//str[0] 不对
//str.charAt(0) => h

```

泛型:

### 自定义泛型

- 自定义泛型接口
  - 基本语法
 

```
interface 接口名<T, R...> {
    •
}
```
  - 注意细节
    - 1) 接口中，静态成员也不能使用泛型(这个和泛型类规定一样)
    - 2) 泛型接口的类型，在继承接口或者实现接口时确定
    - 3) 没有指定类型，默认为Object

## 自定义泛型

### 自定义泛型方法

#### > 基本语法

修饰符 <T,R...> 返回类型 方法名(参数列表) {  
}

#### > 注意细节

1. 泛型方法，可以定义在普通类中，也可以定义在泛型类中
2. 当泛型方法被调用时，类型会确定
3. `public void eat(E e) {}`，修饰符后没有<T,R...> `eat`方法不是泛型方法，而是使用了泛型

> 应用案例 **CustomMethodGeneric.java**  
看老师演示。

```
class Bird<T,R,M>{  
    public<E> void fly(E t){  
        System.out.println(t);  
    }  
}  
  
class Fish {  
    public<A> A show(A t){  
        System.out.println("t的值: "+t);  
        System.out.println("t的类型: " +  
            t.getClass().getSimpleName());  
        return t;  
    }  
}
```

韩顺平  
教育

```
//泛型方法，可以定义在普通类中，也可以定义在泛型类中  
class Car { //普通类  
  
    public void run() { //普通方法  
    }  
  
    //说明 泛型方法  
    //1. <T,R> 就是泛型  
    //2. 是提供给 fly使用的  
    public <T, R> void fly(T t, R r) { //泛型方法  
    }  
}
```

### 泛型的继承和通配符说明 **GenericExtends.java**

#### 1) 泛型不具备继承性

`List<Object> list = new ArrayList<String>();` // 对吗?

#### 2) <?> : 支持任意泛型类型

#### 3) <? extends A> : 支持A类以及A类的子类，规定了泛型的上限

#### 4) <? super A> : 支持A类以及A类的父类，不限于直接父类，规定了泛型的下限

#### 应用案例

```
class AA {  
}  
class BB extends AA {  
}  
class CC extends BB {  
}
```

```
public static void printCollection1(List<?> c) {  
    for (Object object : c) { // 通配符，取出时，就是Object  
        System.out.println(object);  
    }  
}  
// ? extends AA 表示 上限，可以接受 AA或者AA子类  
public static void printCollection2(List<? extends AA> c) {  
    for (Object object : c) {  
        System.out.println(object);  
    }  
}  
// ? super AA 表示 下限，可以接受 AA或者AA父类，不限于直接父类  
public static void printCollection3(List<? super AA> c) {  
    for (Object object : c) {  
        System.out.println(object);  
    }  
}
```

```
ArrayList<Object> a1 = new ArrayList<Object>();  
ArrayList<String> a2 = new ArrayList<String>();  
ArrayList<AA> a3 = new ArrayList<AA>();  
ArrayList<BB> a4 = new ArrayList<BB>();  
ArrayList<CC> a5 = new ArrayList<CC>();
```

韩顺平  
教育

## ● 为什么需要JUnit

1. 一个类有很多功能代码需要测试，为了测试，就需要写入到main方法中
2. 如果有多个功能代码测试，就需要来回注~~销~~，切换很麻烦
3. 如果可以直接运行一个方法，就方便很多，并且可以给出相关信息，就好了 -> JUnit



## ● 基本介绍

1. JUnit是一个Java语言的单元测试框架
2. 多数Java的开发环境都已经集成了JUnit作为单元测试的工具

## ● 使用步骤,看老师演示 JUnit\_java

## ● 包装类和基本数据的转换

演示 包装类 和 基本数据类型的相互转换, 这里以int 和 Integer演示。

- 1) jdk5 前的手动装箱和拆箱方式，装箱: 基本类型->包装类型, 反之，拆箱
- 2) jdk5 以后(含jdk5) 的自动装箱和拆箱方式
- 3) 自动装箱底层调用的是valueOf方法，比如Integer.valueOf()

## ● 案例演示 Integer01.java

```
// 基本类型——>包装类型【手动装箱】
int i = 10;
Integer i1 = new Integer(i);
Integer i2 = Integer.valueOf(i);

// 包装类型——>基本类型【手动拆箱】
Integer j = new Integer(99);
int j1 = j.intValue();
```

// -----jdk5.0之后的方式-----

```
int m = 10;
Integer m2 = m;
Integer n = new Integer(99);
int n2 = n;
System.out.println(n + 100);
System.out.println(n * 2);
if (n > 10) {
}
```



下面的代码是否正确, 为什么?

```
Double d = 100d; //ok, 自动装箱 Double.valueOf(100d);  
Float f = 1.5f; //ok, 自动装箱 Float.valueOf(1.5f);
```

如下两个题目输出结果相同吗? 各是什么?

```
Object obj1 = true? new Integer(1) : new Double(2.0); //三元运算符【是一个整体】 一真  
大师  
System.out.println(obj1); // 什么? 1.0
```

```
Object obj2;  
if(true)  
    obj2 = new Integer(1);  
else  
    obj2 = new Double(2.0);  
System.out.println(obj2);  
输出什么?
```

## String 的常见方法

- equals // 区分大小写, 判断内容是否相等
- equalsIgnoreCase // 忽略大小写的判断内容是否相等
- length // 获取字符的个数, 字符串的长度
- indexOf // 获取字符在字符串中第1次出现的索引, 索引从0开始, 如果找不到, 返回-1
- lastIndexOf // 获取字符在字符串中最后1次出现的索引, 索引从0开始, 如找不到, 返回-1
- substring // 截取指定范围的子串
- trim // 去前后空格
- charAt: 获取某索引处的字符, 注意不能使用Str[index] 这种方式.

看老师的案例演示第二组String相关的方法:

- toUpperCase
- toLowerCase
- concat
- replace 替换字符串中的字符
- split 分割字符串, 对于某些分割字符, 我们需要 转义比如 | \ 等  
案例: String poem = "锄禾日当午, 汗滴禾下土, 谁知盘中餐, 粒粒皆辛苦"; 和 文件路径.
- compareTo // 比较两个字符串的大小
- toCharArray // 转换成字符数组
- format // 格式字符串, %s 字符串 %c 字符 %d 整型 %.2f 浮点型  
案例, 将一个人的信息格式化输出.

集合:

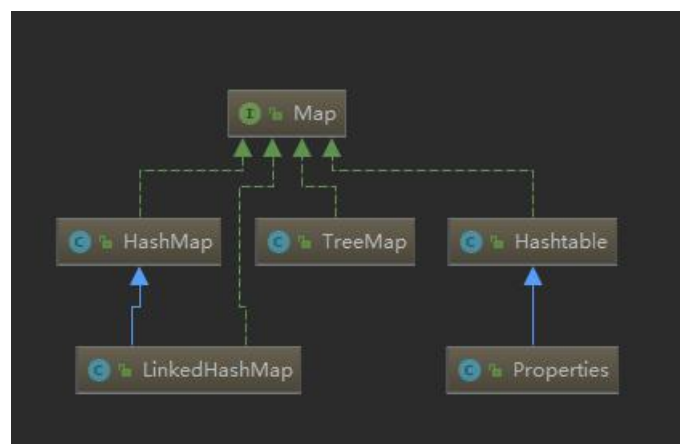
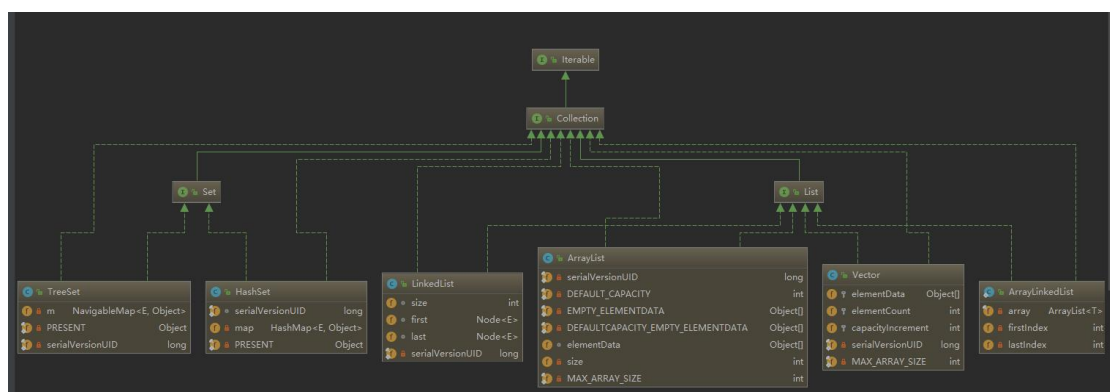
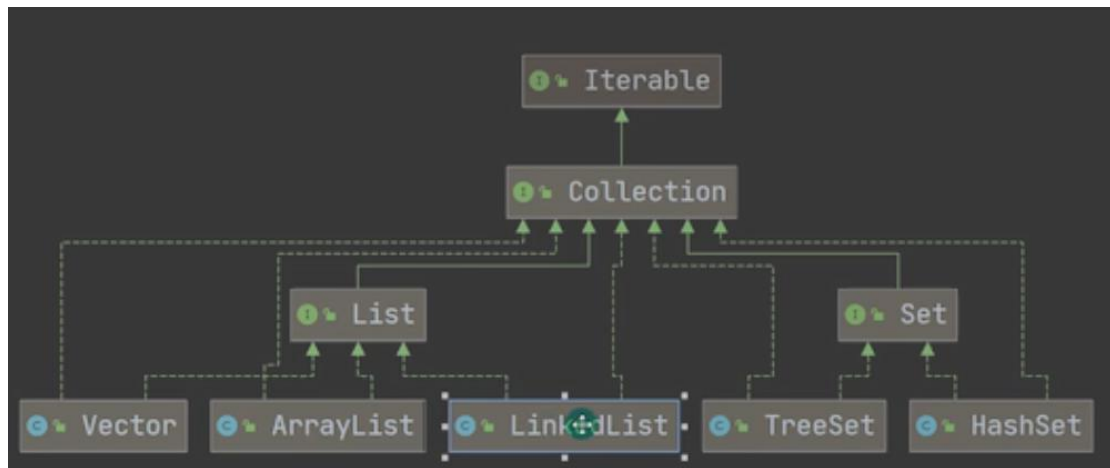
其中数组: 1) 长度开始时必须制定, 且一旦指定便不能修改; 2) 保存的必须为同一类型的元素; 3) 使用数组进行增加元素的示意代码比较麻烦

相较于数组: 集合:

- 1) 可以动态保存任意多个元素(对象)(类型任意), 使用比较方便
- 2) 提供了一系列方便的操作对象方法: add、remove、set、get 等
- 3) 使用集合添加、删除新元素的示意代码比较方便

集合框架图:





集合主要分为两组：

1. 单列集合（单个的对象）
2. 双列结合（键值对形式的）

Collection 接口下有两个重要的子接口 List Set 他们的实现子类都是单列集合

```

ArrayList arrayList = new ArrayList();
arrayList.add("jack");
arrayList.add("tom");
  
```

Map 接口的实现子类的双列集合 存放的 K -V

```
HashMap hashMap = new HashMap();  
hashMap.put("N01", "北京");  
hashMap.put("N02", "上海");
```

Collection 接口实现类的特点:

- 1) collection实现子类可以存放多个元素, 每个元素可以是Object
- 2) 有些Collection的实现类, 可以存放重复的元素, 有些不可以
- 3) 有些Collection的实现类, 有些是有序的(List), 有些不是有序(Set)
- 4) Collection接口没有直接的实现子类, 是通过它的子接口Set 和 List 来实现的

Collection 的常用方法: (ArrayList)

- 1) add:添加单个元素
- 2) remove:删除指定元素
- 3) contains:查找元素是否存在
- 4) size:获取元素个数
- 5) isEmpty:判断是否为空
- 6) clear:清空
- 7) addAll:添加多个元素
- 8) containsAll:查找多个元素是否都存在
- 9) removeAll: 删除多个元素
- 10)说明: 以ArrayList实现类来演示.

Collection 接口遍历元素的方式--->使用 Iterator (迭代器)

- 1) Iterator对象称为迭代器, 主要用于遍历 Collection 集合中的元素。
- 2) 所有实现了Collection接口的集合类都有一个iterator()方法, 用以返回一个实现了Iterator接口的对象, 即可以返回一个迭代器。
- 3) Iterator 的结构.[图:]
- 4) Iterator 仅用于遍历集合, Iterator 本身并不存放对象。

```

Iterator iterator = coll.iterator(); //得到一个集合的迭代器
//hasNext():判断是否还有下一个元素
while(iterator.hasNext()){
//next()作用:1.下移 2.将下移以后集合位置上的元素返回
System.out.println(iterator.next());
}

```

注意：在调用 `iterator.next()` 方法之前必须要调用 `iterator.hasNext()` 进行检测。若不调用且下一条记录无效，则直接调用 `it.next()` 会抛出 `NoSuchElementException` 异常

快速生产成本 while 循环---->itit

List 接口和常用方法：List 接口是 Collection 接口的子接口

List 接口是 Collection 接口的子接口 **List .java**

- 1) List 集合类中元素有序(即添加顺序和取出顺序一致)、且可重复 [案例]
- 2) List 集合中的每个元素都有其对应的顺序索引，即支持索引。 [案例]
- 3) List 容器中的元素都对应一个整数型的序号记载其在容器中的位置，可以根据序号存取容器中的元素。
- 4) JDK API 中 List 接口的实现类有：

```

java.util
接口 List<E>
所有超接口:
Collection<E>, Iterable<E>
所有已实现类:
AbstractList, AbstractSequentialList, ArrayList, ArrayDeque,
ConcurrentLinkedQueue, LinkedList, ListIterator, RandomAccess,
Stack, Vector

```

常用的有：ArrayList、LinkedList 和 Vector。

```

List list = new ArrayList();
list.add("tom");
list.add("jack");
list.add("mary");
list.add("mary");
list.add("smith2");
list.add("kristina");
System.out.println(list);

```

3的案例:  
System.out.println(list.get(4)); //smith2



## List 集合里添加了一些根据索引来操作集合元素的方法

- 1) void add(int index, Object ele):在index位置插入ele元素
- 2) boolean addAll(int index, Collection eles):从index位置开始将eles中的所有元素添加进来
- 3) Object get(int index):获取指定index位置的元素
- 4) int indexOf(Object obj):返回obj在集合中首次出现的位置
- 5) int lastIndexOf(Object obj):返回obj在当前集合中末次出现的位置
- 6) Object remove(int index):移除指定index位置的元素, 并返回此元素
- 7) Object set(int index, Object ele):设置指定index位置的元素为ele, 相当于是替换.
- 8) List subList(int fromIndex, int toIndex):返回从fromIndex到toIndex位置的子集合

List 的三种遍历方式:

## • List的三种遍历方式 [ArrayList, LinkedList, Vector]

### ListFor.java

- 1) 方式一: 使用iterator

```
Iterator iter = col.iterator();  
while(iter.hasNext()){  
    Object o = iter.next();  
}
```

- 2) 方式二: 使用增强for

```
for(Object o:col){  
}
```

- 3) 方式三: 使用普通for

```
for(int i=0;i<list.size();i++){  
    Object object = list.get(i);  
    System.out.println(object);  
}
```

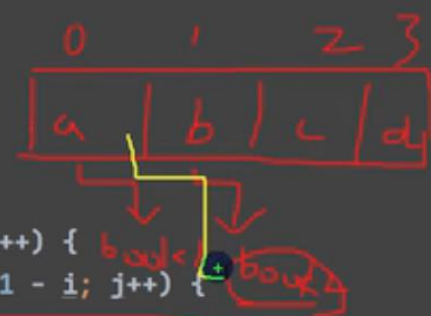
说明: 使用LinkedList完成 使用方式和 ArrayList 一样

```
List list = new LinkedList();  
list.add("tom");  
list.add("jack");  
list.add("smith");  
list.add("vernasa");  
list.add("kristina");
```

```
Iterator iterator = list.iterator();  
while(iterator.hasNext){  
    Object obj = iterator.next();  
    System.out.println("obj = " + obj);  
}
```

```
//静态方法
//价格要求是从小到大
public static void sort(List list) {

    int listSize = list.size();
    for (int i = 0; i < listSize - 1; i++) {
        for (int j = 0; j < listSize - 1 - i; j++) {
            //取出对象Book
            Book book1 = (Book) list.get(j);
            Book book2 = (Book) list.get(j + 1);
            if (book1.getPrice() > book2.getPrice()) { //交换
                list.set(j, book2);
                list.set(j + 1, book1);
            }
        }
    }
}
```



ArrayList 使用的注意事项:

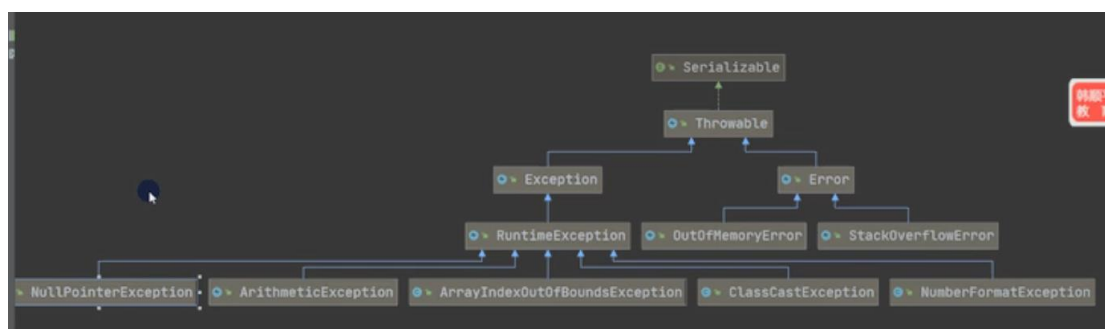
### ArrayListDetail.java

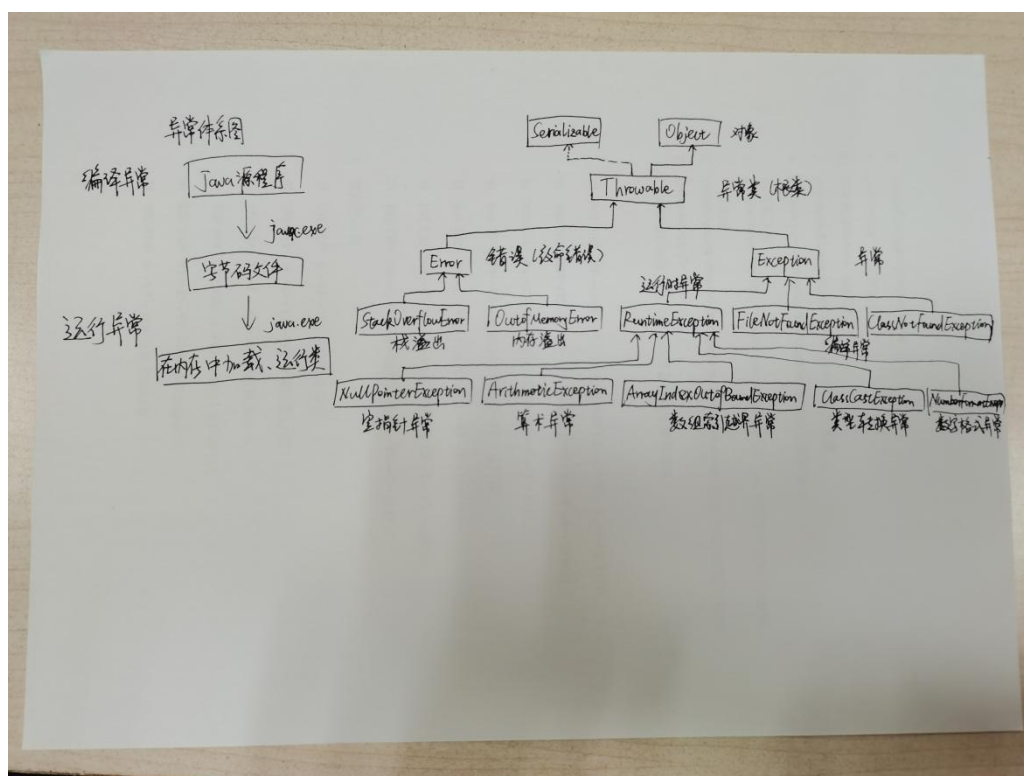
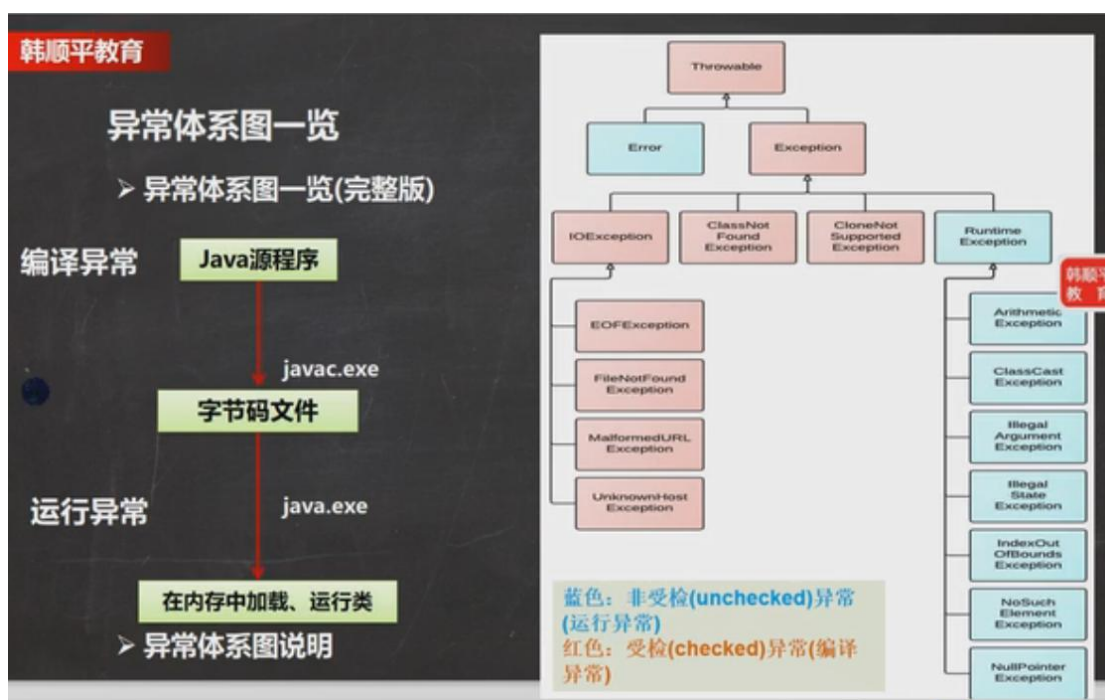
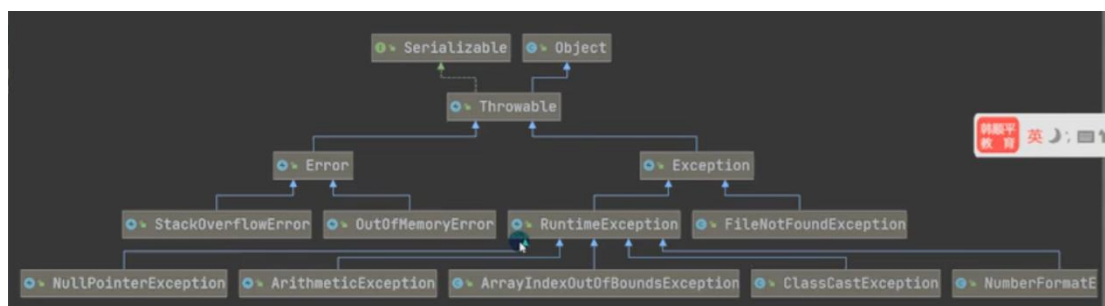
- 1) permits all elements, including null, ArrayList 可以加入null,并且多个
- 2) ArrayList 是由数组来实现数据存储的
- 3) ArrayList 基本等同于Vector, 除了 ArrayList是线程不安全(执行效率高) 看源码. , 在多线程情况下, 不建议使用ArrayList

### ArrayList的底层操作机制源码分析(重点, 难点.)

ArrayListSource.java, 先说结论, 在分析源码(示意图)

- 1) ArrayList中维护了一个Object类型的数组elementData. [debug 看源码]  
transient Object[] elementData; //transient 表示瞬间,短暂的, 表示该属性不会被序列化
- 2) 当创建ArrayList对象时, 如果使用的是无参构造器, 则初始elementData容量为0, 第1次添加, 则扩容elementData为10, 如需要再次扩容, 则扩容elementData为1.5倍。
- 3) 如果使用的是指定大小的构造器, 则初始elementData容量为指定大小, 如果需要扩容, 则直接扩容elementData为1.5倍。







Vector 的基本介绍:

- 1) Vector 类的定义说明;
- 2) Vector 底层也是一个对象数组, protected Object[] elementData;
- 3) Vector 是线程同步的, 即线程安全, Vector 类的操作方法带有 synchronized;

```
public synchronized E get(int index){  
    If (index >= elementCount){  
        throw new ArrayIndexOutOfBoundsException(index);  
        Return elementData(index);  
    }  
}
```

- 4) 在开发中, 需要线程同步安全时, 考虑使用 Vector

Vector 和 ArrayList 的比较:

	底层结构	版本	线程安全 (同步) 效率	扩容倍数
ArrayList	可变数组	jdk1.2	不安全, 效率高	如果有参构造1.5倍 如果是无参 1.第一次10 2.从第二次开始按1.5扩
Vector	可变数组	jdk1.0	安全, 效率不高	如果是无参, 默认10 , 满后, 就按2倍扩容  如果指定大小, 则每次直接按2倍扩

LinkedList 底层结构:

- 1) LinkedList 实现了双向链表和双端队列特点;
- 2) 可以添加任意元素 (元素可以重复), 包括 null;
- 3) 线程不安全, 没有实现同步。

LinkedList 的底层操作机制:

- 1) LinkedList 底层维护了一个双向列表;
- 2) LinkedList 中维护了两个属性 first 和 last 分别指向首节点和尾节点;
- 3) 每个结点 (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, 通过 next 指向后一个结点。最终实现双向列表;
- 4) 所以 linkedList 的元素的添加和删除, 不是通过数组完成的, 相对来说效率较高;
- 5) 模拟一个简单的双向链表。

### ● LinkedList的增删改查案例

#### LinkedListCRUD.java

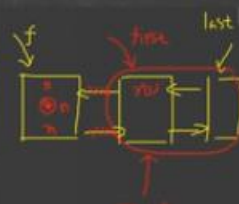
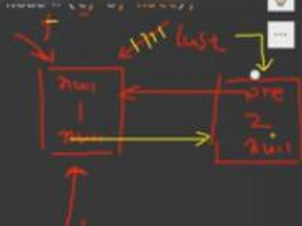
```

LinkedList linkedList = new LinkedList();
for(int i = 1; i <= 2; i++) {
    linkedList.add(i);
}
linkedList.add(100);
linkedList.add(100);
for (Object object : linkedList) {
    System.out.println(object);
}
//linkedList.remove(0);
//linkedList.remove(kk);

linkedList.set(0, "韩顺平教育");
System.out.println("==" );
for (Object object : linkedList) {
    System.out.println(object);
}
Object object = linkedList.get(0);
System.out.println("object=" + object);
System.out.println(linkedList.getFirst());

```

给同学们debug LinkedList的add底层源码

ArrayList 和 LinkedList 的比较

	底层结构	增删的效率	改查的效率
ArrayList	可变数组	较低 数组扩容	较高
LinkedList	双向链表	较高, 通过链表追加	较低

如何选择ArrayList和LinkedList:

- 1) 如果我们改查的操作多, 选择ArrayList
- 2) 如果我们增删的操作多, 选择LinkedList
- 3) 一般来说, 在程序中, 80%-90%都是查询, 因此大部分情况下会选择ArrayList
- 4) 在一个项目中, 根据业务灵活选择, 也可能这样, 一个模块使用的是ArrayList, 另外一个模块是LinkedList.

Set 接口基本介绍:

- 1) 无序 (添加和去除的顺序不一致), 没有索引
- 2) 不允许重复元素, 所以最多包含一个 null
- 3) JDK API 中 Set 接口的实现类包括:

```

java.util
接口 Set<E>
类型参数:
E - 此 set 所维护元素的类型
所有超级接口:
Collection<E>, Iterable<E>
所有已知子接口:
NavigableSet<E>, SortedSet<E>
所有已知实现类:
AbstractSet, ConcurrentSkipListSet, CopyOnWriteArraySet, EnumSet, HashSet,
JConcurrentHashSet, LinkedHashSet, TreeSet

```

Set 接口的常用方法:

和 List 接口一样, Set 接口也是 Collection 的子接口, 因此常用方法和 Collection

接口一样。

Set 接口的遍历方式：

同 Collection 的遍历方式一样，因为 Set 接口是 Collection 接口的子接口：

1. 可以使用迭代器 `Iterator iterator = iterator.iterator();`
2. 增强 for 循环
3. 不能使用索引的方式来获取

```
//老韩解读
//1. 以Set 接口的实现类 HashSet 来讲解Set 接口的方法
Set set = new HashSet();
set.add("john");
set.add("lucy");
set.add("john");//重复
set.add("jack");
set.add(null);
set.add(null);
```

```
//老韩解读
//1. 以Set 接口的实现类 HashSet 来讲解Set 接口的方法
//2. set 接口的实现类的对象(Set接口对象)，不能存放重复的元素，可以添加一个null
//3. set 接口对象存放数据是无序(即添加的顺序和取出的顺序不一致)
Set set = new HashSet();
set.add("john");
set.add("lucy");
set.add("john");//重复
set.add("jack");
set.add(null);
set.add(null);//再次添加null
```

取出的顺序虽然不是添加的顺序，但是是固定的。