=======================第一节课开始=============================================

###01集合使用的回顾

\*A:集合使用的回顾

\*a.ArrayList集合存储5个int类型元素

public static void main(String[] args) {

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

list.add(111);

list.add(222);

list.add(333);

list.add(444);

list.add(555);

for(int i=0; i<list.size(); i++){

System.out.println(list.get(i));

}

}

\*b.ArrayList集合存储5个Person类型元素

public static void main(String[] args) {

ArrayList<Person> list = new ArrayList<Person>();

list.add(new Person(“小强”));

list.add(new Person(“老王”));

list.add(new Person(“小虎”));

list.add(new Person(“小泽”));

list.add(new Person(“小红”));

for(int i=0; i<list.size(); i++){

Person p = list.get(i);

System.out.println(p);

}

}

###02集合的学习目标

集合，集合是java中提供的一种容器，可以用来存储多个数据。

在前面的学习中，我们知道数据多了，可以使用数组存放或者使用ArrayList集合进行存放数据。那么，集合和数组既然都是容器，它们有啥区别呢？

 数组的长度是固定的。集合的长度是可变的。

 集合中存储的元素必须是引用类型数据

###03集合继承关系图

A:集合继承关系图

a:ArrayList的继承关系:

查看ArrayList类发现它继承了抽象类AbstractList同时实现接口List，而List接口又继承了Collection接口。Collection接口为最顶层集合接口了。

源代码：

interface List extends Collection {

}

public class ArrayList extends AbstractList implements List{

}

b:集合继承体系

这说明我们在使用ArrayList类时，该类已经把所有抽象方法进行了重写。那么，实现Collection接口的所有子类都会进行方法重写。

 Collection接口常用的子接口有：List接口、Set接口

 List接口常用的子类有：ArrayList类、LinkedList类

 Set接口常用的子类有：HashSet类、LinkedHashSet类

Collection 接口

|

----------------------------------------------------------------

| |

List接口 Set接口

| |

---------------- -------------

| | | |

ArrayList类 LinkedList类 HashSet类 LinkedHashSet类

###04集合Collection的方法

A:集合Collection的方法

/\*

\* Collection接口中的方法

\* 是集合中所有实现类必须拥有的方法

\* 使用Collection接口的实现类,程序的演示

\* ArrayList implements List

\* List extends Collection

\* 方法的执行,都是实现的重写

\*/

public class CollectionDemo {

public static void main(String[] args) {

function\_2();

}

/\* Collection接口方法

\* Object[] toArray() 集合中的元素,转成一个数组中的元素, 集合转成数组

\* 返回是一个存储对象的数组, 数组存储的数据类型是Object

\*/

private static void function\_2() {

Collection<String> coll = new ArrayList<String>();

coll.add("abc");

coll.add("itcast");

coll.add("itheima");

coll.add("money");

coll.add("123");

Object[] objs = coll.toArray();

for(int i = 0 ; i < objs.length ; i++){

System.out.println(objs[i]);

}

}

/\*

\* 学习Java中三种长度表现形式

\* 数组.length , 属性 返回值 int

\* 字符串.length() , 方法,返回值int

\* 集合.size() , 方法, 返回值int

\*/

/\*

\* Collection接口方法

\* boolean contains(Object o) 判断对象是否存在于集合中,对象存在返回true

\* 方法参数是Object类型

\*/

private static void function\_1() {

Collection<String> coll = new ArrayList<String>();

coll.add("abc");

coll.add("itcast");

coll.add("itheima");

coll.add("money");

coll.add("123");

boolean b = coll.contains("itcast");

System.out.println(b);

}

/\*

\* Collection接口的方法

\* void clear() 清空集合中的所有元素

\* 集合容器本身依然存在

\*/

public static void function(){

//接口多态的方式调用

Collection<String> coll = new ArrayList<String>();

coll.add("abc");

coll.add("bcd");

System.out.println(coll);

coll.clear();

System.out.println(coll);

}

}

###05集合Collection的remove方法

A:05集合Collection的remove方法

/\*

\* Collection接口方法

\* boolean remove(Object o)移除集合中指定的元素 ,删除第一遇到的相同元素，如果集合中有两个这样的元素，会删除先出现的那一个。

\*/

private static void function\_3(){

Collection<String> coll = new ArrayList<String>();

coll.add("abc");

coll.add("money");

coll.add("itcast");

coll.add("itheima");

coll.add("money");

coll.add("123");

System.out.println(coll);

boolean b = coll.remove("money"); //删除第一个“money”，后面那一个不会删除，将会保留下来

System.out.println(b);

System.out.println(coll)

}

=======================第二节课开始=============================================

###06迭代器的概述

A:迭代器概述:

a:java中提供了很多个集合，它们在存储元素时，采用的存储方式不同。

我们要取出这些集合中的元素，可通过一种通用的获取方式来完成。

b:Collection集合元素的通用获取方式：在取元素之前先要判断集合中有没有元素，

如果有，就把这个元素取出来，继续在判断，如果还有就再取出出来。一直把集合中的所有元素全部取出。这种取出方式专业术语称为迭代。

c:每种集合的底层的数据结构不同,例如ArrayList是数组,LinkedList底层是链表,但是无论使用那种集合,我们都会有判断是否有元素

以及取出里面的元素的动作,那么Java为我们提供一个迭代器定义了统一的判断元素和取元素的方法

###07迭代器的实现原理

\*A:迭代器的实现原理

/\*

\* 集合中的迭代器:

\* 获取集合中元素方式

\* 接口 Iterator : 两个抽象方法

\* boolean hasNext() 判断集合中还有没有可以被取出的元素,如果有返回true

\* next() 取出集合中的下一个元素

\*

\* Iterator接口,找实现类.

\* Collection接口定义方法

\* Iterator iterator()

\* ArrayList 重写方法 iterator(),返回了Iterator接口的实现类的对象

\* 使用ArrayList集合的对象

\* Iterator it =array.iterator(),运行结果就是Iterator接口的实现类的对象

\* it是接口的实现类对象,调用方法 hasNext 和 next 集合元素迭代

\*/

###08迭代器的代码实现

\*A:迭代器的代码实现

public class IteratorDemo {

public static void main(String[] args) {

Collection<String> coll = new ArrayList<String>();

coll.add("abc1");

coll.add("abc2");

coll.add("abc3");

coll.add("abc4");

//迭代器,对集合ArrayList中的元素进行取出

//调用集合的方法iterator()获取出,Iterator接口的实现类的对象

Iterator<String> it = coll.iterator();

//接口实现类对象,调用方法hasNext()判断集合中是否有元素

//boolean b = it.hasNext();

//System.out.println(b);

//接口的实现类对象,调用方法next()取出集合中的元素

//String s = it.next();

//System.out.println(s);

//迭代是反复内容,使用循环实现,循环的条件,集合中没元素, hasNext()返回了false

while(it.hasNext()){

String s = it.next();

System.out.println(s);

}

}

}

###09迭代器的执行过程

A:迭代器的执行过程

a:迭代器的原理:

while(it.hasNext()) {

System.out.println(it.next());

}

//cursor记录的索引值不等于集合的长度返回true,否则返回false

public boolean hasNext() {

return cursor != size; //cursor初值为0

}

//next()方法作用:

//①返回cursor指向的当前元素

//②cursor++

public Object next() {

int i = cursor;

cursor = i + 1;

return elementData[lastRet = i];

}

b:for循环迭代写法:

for (Iterator<String> it2 = coll.iterator(); it2.hasNext(); ) {

System.out.println(it2.next());

}

###10集合迭代中的转型

A:集合迭代中的转型

a:在使用集合时，我们需要注意以下几点：

 集合中存储其实都是对象的地址。

 集合中可以存储基本数值吗？jdk1.5版本以后可以存储了。

因为出现了基本类型包装类，它提供了自动装箱操作（基本类型对象），这样，集合中的元素就是基本数值的包装类对象。

b:存储时提升了Object。取出时要使用元素的特有内容，必须向下转型。

Collection coll = new ArrayList();

coll.add("abc");

coll.add("aabbcc");

coll.add("shitcast");

Iterator it = coll.iterator();

while (it.hasNext()) {

//由于元素被存放进集合后全部被提升为Object类型

//当需要使用子类对象特有方法时，需要向下转型

String str = (String) it.next();

System.out.println(str.length());

}

注意：如果集合中存放的是多个对象，这时进行向下转型会发生类型转换异常。

c:Iterator接口也可以使用<>来控制迭代元素的类型的。代码演示如下：

Collection<String> coll = new ArrayList<String>();

coll.add("abc");

coll.add("aabbcc");

coll.add("shitcast");

Iterator<String> it = coll.iterator();

while (it.hasNext()) {

String str = it.next();

//当使用Iterator<String>控制元素类型后，就不需要强转了。获取到的元素直接就是String类型

System.out.println(str.length());

}

=========================第三节课开始====================================

###11增强for循环遍历数组

\*A:增强for循环遍历数组

a:格式:

/\*

\* JDK1.5新特性,增强for循环

\* JDK1.5版本后,出现新的接口 java.lang.Iterable

\* Collection开是继承Iterable

\* Iterable作用,实现增强for循环

\*

\* 格式:

\* for( 数据类型 变量名 : 数组或者集合 ){

\* sop(变量);

\* }

\*/

public static void function\_1(){

//for对于对象数组遍历的时候,能否调用对象的方法呢

String[] str = {"abc","itcast","cn"};

for(String s : str){

System.out.println(s.length());

}

}

/\*

\* 实现for循环,遍历数组

\* 好处: 代码少了,方便对容器遍历

\* 弊端: 没有索引,不能操作容器里面的元素

\*/

public static void function(){

int[] arr = {3,1,9,0};

for(int i : arr){

System.out.println(i+1);

}

System.out.println(arr[0]);

}

###12增强for循环遍历集合

A:增强for循环遍历集合

/\*

\* 增强for循环遍历集合

\* 存储自定义Person类型

\*/

public static void function\_2(){

ArrayList<Person> array = new ArrayList<Person>();

array.add(new Person("a",20));

array.add(new Person("b",10));

for(Person p : array){

System.out.println(p);// System.out.println(p.toString());

}

}

###13泛型的引入

A:泛型的引入

在前面学习集合时，我们都知道集合中是可以存放任意对象的，

只要把对象存储集合后，那么这时他们都会被提升成Object类型。

当我们在取出每一个对象，并且进行相应的操作，这时必须采用类型转换。比如下面程序：

public class GenericDemo {

public static void main(String[] args) {

List list = new ArrayList();

list.add("abc");

list.add("itcast");

list.add(5);//由于集合没有做任何限定，任何类型都可以给其中存放

//相当于:Object obj=new Integer(5);

Iterator it = list.iterator();

while(it.hasNext()){

//需要打印每个字符串的长度,就要把迭代出来的对象转成String类型

String str = (String) it.next();//String str=(String)obj;

//编译时期仅检查语法错误,String是Object的儿子可以向下转型

//运行时期String str=(String)(new Integer(5))

//String与Integer没有父子关系所以转换失败

//程序在运行时发生了问题java.lang.ClassCastException

System.out.println(str.length());

}

}

}

###14泛型的定义和使用

A:泛型的定义和使用

/\*

\* JDK1.5 出现新的安全机制,保证程序的安全性

\* 泛型: 指明了集合中存储数据的类型 <数据类型>

\*/

public class GenericDemo {

public static void main(String[] args) {

function();

}

public static void function(){

Collection<String> coll = new ArrayList<String>();

coll.add("abc");

coll.add("rtyg");

coll.add("43rt5yhju");

// coll.add(1);

Iterator<String> it = coll.iterator();

while(it.hasNext()){

String s = it.next();

System.out.println(s.length());

}

}

}

###15Java中的伪泛型

A:Java中的伪泛型：

泛型只在编译时存在,编译后就被擦除,在编译之前我们就可以限制集合的类型,起到作用

例如:ArrayList<String> al=new ArrayList<String>();

编译后:ArrayList al=new ArrayList();

================================第四节课开始======================================================

###16泛型类

A:泛型类:

a:定义格式：

修饰符 class 类名<代表泛型的变量> { }

例如，API中的ArrayList集合：

class ArrayList<E>{

public boolean add(E e){ }

public E get(int index){ }

}

b:使用格式：

创建对象时，确定泛型的类型

例如，ArrayList<String> list = new ArrayList<String>();

此时，变量E的值就是String类型

class ArrayList<String>{

public boolean add(String e){ }

public String get(int index){ }

}

例如，ArrayList<Integer> list = new ArrayList<Integer>();

此时，变量E的值就是Integer类型

class ArrayList<Integer>{

public boolean add(Integer e){ }

public Integer get(int index){ }

}

###17泛型的方法

A:泛型的方法

a:定义格式：修饰符 <代表泛型的变量> 返回值类型 方法名(参数){ }

b:泛型方法的使用:

1:例如，API中的ArrayList集合中的方法：

public <T> T[] toArray(T[] a){ }

//该方法，用来把集合元素存储到指定数据类型的数组中，返回已存储集合元素的数组

使用格式：调用方法时，确定泛型的类型

例如:

ArrayList<String> list = new ArrayList<String>();

String[] arr = new String[100];

String[] result = list.toArray(arr);

此时，变量T的值就是String类型。变量T，可以与定义集合的泛型不同

public <String> String[] toArray(String[] a){ }



例如:

ArrayList<String> list = new ArrayList<String>();

Integer[] arr = new Integer[100];

Integer [] result = list.toArray(arr);

此时，变量T的值就是Integer类型。变量T，可以与定义集合的泛型不同

public <Integer> Integer[] toArray(Integer[] a){ }

###18泛型的接口

A:泛型的接口:

/\*

\* 带有泛型的接口

\*

\* public interface List <E>{

\* abstract boolean add(E e);

\* }

\*

\* 实现类,先实现接口,不理会泛型

\* public class ArrayList<E> implements List<E>{

\* }

\* 调用者 : new ArrayList<String>() 后期创建集合对象的时候,指定数据类型

\*

\*

\* 实现类,实现接口的同时,也指定了数据类型

\* public class XXX implements List<String>{

\* }

\* new XXX()

\*/

public class GenericDemo2 {

}

###19泛型的好处

A:泛型的好处

a:将运行时期的ClassCastException，转移到了编译时期变成了编译失败。

b:避免了类型强转的麻烦。

演示下列代码：

public class GenericDemo {

public static void main(String[] args) {

List<String> list = new ArrayList<String>();

list.add("abc");

list.add("itcast");

//list.add(5);//当集合明确类型后，存放类型不一致就会编译报错

//集合已经明确具体存放的元素类型，那么在使用迭代器的时候，迭代器也同样会知道具体遍历元素类型

Iterator<String> it = list.iterator();

while(it.hasNext()){

String str = it.next();

System.out.println(str.length()); //当使用Iterator<String>

//控制元素类型后，就不需要强转了。获取到的元素直接就是String类型

}

}

}

###20泛型的通配符

A:泛型的通配符

/\*

\* 泛型的通配符

\*/

public class GenericDemo {

public static void main(String[] args) {

ArrayList<String> array = new ArrayList<String>();

HashSet<Integer> set = new HashSet<Integer>();

array.add("123");

array.add("456");

set.add(789);

set.add(890);

iterator(array);

iterator(set);

}

/\*

\* 定义方法,可以同时迭代2个集合

\* 参数: 怎么实现 , 不能写ArrayList,也不能写HashSet

\* 参数: 或者共同实现的接口

\* 泛型的通配,匹配所有的数据类型 ?

\*/

public static void iterator(Collection<?> coll){

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

while(it.hasNext()){

//it.next()获取的对象,什么类型

System.out.println(it.next());

}

}

}

###21泛型的限定

A:泛型的限定

/\*

\* 将的酒店员工,厨师,服务员,经理,分别存储到3个集合中

\* 定义方法,可以同时遍历3集合,遍历三个集合的同时,可以调用工作方法

\*/

import java.util.ArrayList;

import java.util.Iterator;

public class GenericTest {

public static void main(String[] args) {

//创建3个集合对象

ArrayList<ChuShi> cs = new ArrayList<ChuShi>();

ArrayList<FuWuYuan> fwy = new ArrayList<FuWuYuan>();

ArrayList<JingLi> jl = new ArrayList<JingLi>();

//每个集合存储自己的元素

cs.add(new ChuShi("张三", "后厨001"));

cs.add(new ChuShi("李四", "后厨002"));

fwy.add(new FuWuYuan("翠花", "服务部001"));

fwy.add(new FuWuYuan("酸菜", "服务部002"));

jl.add(new JingLi("小名", "董事会001", 123456789.32));

jl.add(new JingLi("小强", "董事会002", 123456789.33));

// ArrayList<String> arrayString = new ArrayList<String>();

iterator(jl);

iterator(fwy);

iterator(cs);

}

/\*

\* 定义方法,可以同时遍历3集合,遍历三个集合的同时,可以调用工作方法 work

\* ? 通配符,迭代器it.next()方法取出来的是Object类型,怎么调用work方法

\* 强制转换: it.next()=Object o ==> Employee

\* 方法参数: 控制,可以传递Employee对象,也可以传递Employee的子类的对象

\* 泛型的限定 本案例,父类固定Employee,但是子类可以无限?

\* ? extends Employee 限制的是父类, 上限限定, 可以传递Employee,传递他的子类对象

\* ? super Employee 限制的是子类, 下限限定, 可以传递Employee,传递他的父类对象

\*/

public static void iterator(ArrayList<? extends Employee> array){

Iterator<? extends Employee> it = array.iterator();

while(it.hasNext()){

//获取出的next() 数据类型,是什么Employee

Employee e = it.next();

e.work();

}

}

}