* 早上：9:30进行测试 10:00正式开始
* 网络环境
* 建议先把音量调到最大
* 调整屏幕布局
  + 客户端菜单 – 布局
* 调成比800\*600更大的分辨率
* 定期讲课
  + http://www.bjsxt.com
* 设计模式之Iterator

**假设**

1. 初步具备面向对象的设计思维
   1. 网络课程第二课
   2. 视频第三章
2. 了解多态的概念
   1. 网络课程第二课
   2. 视频第三章
3. 了解JDK集合类（容器类）（非必须）
   1. 视频第七章
   2. 所有的讲解参见源码
4. 请自己动手写一个可以动态添加对象的容器
5. 考虑容器的可替换性

* 作业
* 自己阅读观察者模式
* <http://www.bjsxt.com/> 视频下载

主要用在容器的遍历

使用数组定义一个容器：

package com.test.designpattern.\_004;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class MyArrayList{  
 private Object[] objects= new Object[16];  
 private int currentIndex=0;  
 private int size=16;  
  
 /\*\*  
 \* @since 1.6  
 \* @param v  
 \* @return  
 \*/  
 public MyArrayList add(Object v){  
 if(this.currentIndex==this.size){  
 System.out.println(this.currentIndex);  
 this.size\*=2;  
// this.objects= copy();  
 Object[] newObjects=new Object[this.size];  
 System.arraycopy(this.objects,0,newObjects,0,this.objects.length);  
 this.objects=newObjects;  
 }  
 System.out.println(this.currentIndex);  
 this.objects[this.currentIndex++]=v;  
 return this;  
 }  
  
 private Object[] copy(){  
 Object[] newObject=new Object[this.size];  
 for(int i=0;i<this.currentIndex;i++){  
 newObject[i]=this.objects[i];  
 }  
  
 return newObject;  
 }  
  
 public static void main(String[] args){  
 MyArrayList myArrayList=new MyArrayList();  
 for(int index=0;index<100;index++){  
 myArrayList.add(index);  
 }  
 }  
}

使用链表实现容器：

package com.test.designpattern.\_004;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class MyArrayList2 {  
 private Node nodeHead;  
 private int size;  
 private Node lastNode;  
  
 public MyArrayList2 add(Object v){  
 Node node=new Node(v,null);  
 if(size==0) {  
 this.nodeHead=node;  
 this.lastNode=node;  
 }else{  
 this.lastNode.next=node;  
 this.lastNode=node;  
 }  
  
 this.size++;  
 return this;  
 }  
  
 private class Node{  
 private Object v;  
 private Node next;  
  
 public Node(Object v,Node n){  
 this.v=v;  
 this.next=n;  
 }  
 }  
  
 public static void main(String[] args) {  
 MyArrayList2 list2=new MyArrayList2();  
 for(int i=0;i<100;i++){  
 list2.add(i);  
 }  
  
 System.out.println(list2.size);  
 Node currentNode=list2.nodeHead;  
 while(true){  
 System.out.println(currentNode.v);  
 if(currentNode.next==null) break;  
 currentNode=currentNode.next;  
 }  
 }  
}

为了统一遍历方式，使用Iterator,完整实现：

package com.test.designpattern.\_004;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/15.  
 \*/  
public interface MyIterator<T> { #迭代器接口  
 boolean hasNext();  
 Object next();  
}

package com.test.designpattern.\_004;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/15.  
 \*/  
public interface IMyArrayList extends MyIterator<IMyArrayList>{  
 IMyArrayList add(Object o);  
 MyIterator<IMyArrayList> iterator(); #产生迭代器方法

}

package com.test.designpattern.\_004;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class MyArrayList implements IMyArrayList{  
 private Object[] objects= new Object[16];  
 private int currentIndex=0;  
 private int size=16;  
  
 private MyIterator<IMyArrayList> myIterator; #持有自身迭代器对象  
  
 /\*\*  
 \* @since 1.6  
 \* @param v  
 \* @return  
 \*/  
 public MyArrayList add(Object v){  
 if(this.currentIndex==this.size){  
 System.out.println(this.currentIndex);  
 this.size\*=2;  
// this.objects= copy();  
 Object[] newObjects=new Object[this.size];  
 System.arraycopy(this.objects,0,newObjects,0,this.objects.length);  
 this.objects=newObjects;  
 }  
 System.out.println(this.currentIndex);  
 this.objects[this.currentIndex++]=v;  
 return this;  
 }  
  
 public MyIterator<IMyArrayList> iterator(){  
 this.myIterator=new MyArraylistIterator(this); #返回内部类定义的迭代器对象，这里可以使用单例  
 return this.myIterator;  
 }  
  
 public boolean hasNext(){ #hasNext方法，代用自身迭代器的hasNext方法  
 return myIterator.hasNext();  
 }  
  
 public Object next(){  
 return myIterator.next();  
 }  
  
 private Object[] copy(){  
 Object[] newObject=new Object[this.size];  
 for(int i=0;i<this.currentIndex;i++){  
 newObject[i]=this.objects[i];  
 }  
  
 return newObject;  
 }  
  
 private class MyArraylistIterator implements MyIterator<IMyArrayList>{ #内部类，迭代器对象  
 private MyArrayList myArrayList; #迭代外部类需要的字段  
 private int cursor;  
  
 public MyArraylistIterator(MyArrayList myArrayList){  
 this.myArrayList=myArrayList;  
 }  
  
 public boolean hasNext() {  
 if(cursor>=this.myArrayList.getCurrentIndex())return false;  
 return true;  
 }  
  
 public Object next() {  
 return this.myArrayList.getObjects()[cursor++];  
 }  
 }  
  
 public Object[] getObjects() {  
 return objects;  
 }  
  
 public void setObjects(Object[] objects) {  
 this.objects = objects;  
 }  
  
 public int getCurrentIndex() {  
 return currentIndex;  
 }  
  
 public void setCurrentIndex(int currentIndex) {  
 this.currentIndex = currentIndex;  
 }  
  
 public int getSize() {  
 return size;  
 }  
  
 public void setSize(int size) {  
 this.size = size;  
 }  
}

package com.test.designpattern.\_004;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class MyArrayList2 implements IMyArrayList{  
 private Node nodeHead;  
 private int size;  
 private Node lastNode;  
  
 private MyArrayList2Iterator myArrayList2Iterator;  
  
 public MyArrayList2 add(Object v){  
 Node node=new Node(v,null);  
 if(size==0) {  
 this.nodeHead=node;  
 this.lastNode=node;  
 }else{  
 this.lastNode.next=node;  
 this.lastNode=node;  
 }  
  
 this.size++;  
 return this;  
 }  
  
 public MyIterator<IMyArrayList> iterator(){#返回内部类定义的迭代器对象，这里可以使用单例  
 this.myArrayList2Iterator=new MyArrayList2Iterator(this);  
 return this.myArrayList2Iterator;  
 }  
 public boolean hasNext(){return true;}  
 public Object next(){return null;}  
  
 private class Node{  
 private Object v;  
 private Node next;  
  
 public Node(Object v,Node n){  
 this.v=v;  
 this.next=n;  
 }  
  
 public Object getV() {  
 return v;  
 }  
  
 public void setV(Object v) {  
 this.v = v;  
 }  
 }  
  
 private class MyArrayList2Iterator implements MyIterator<IMyArrayList>{  
 private MyArrayList2 myArrayList2;  
 private Node currentNode;  
  
 public MyArrayList2Iterator(MyArrayList2 myArrayList2){  
 this.myArrayList2=myArrayList2;  
 this.currentNode=myArrayList2.nodeHead;  
 }  
  
 public boolean hasNext() {  
 return this.currentNode.next!=null;  
 }  
  
 public Object next() {  
 Node thatNode= this.currentNode.next;  
 this.currentNode=this.currentNode.next;  
 return thatNode.getV();  
 }  
 }  
}

测试：

package com.test.designpattern.\_004;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/15.  
 \*/  
public class Main { #测试  
 public static void main(String[] args) {  
// IMyArrayList myArrayList=new MyArrayList(); #使用第一种  
 IMyArrayList myArrayList=new MyArrayList2(); #使用第二种  
 for(int i=0;i<100;i++){  
 myArrayList.add(i);  
 }  
  
 MyIterator<IMyArrayList> iterator=myArrayList.iterator();  
 while(iterator.hasNext()){  
 System.out.println(iterator.next());  
 }  
 }  
}

将以上容器写成泛型形式：

package com.test.designpattern.\_004;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/15.  
 \*/  
public interface MyIterator<T> {  
 boolean hasNext();  
 T next();  
}

package com.test.designpattern.\_004;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/15.  
 \*/  
public interface IMyArrayList<T> extends MyIterator<T>{  
 IMyArrayList<T> add(T o);  
 MyIterator<IMyArrayList<T>> iterator();  
}

package com.test.designpattern.\_004;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class MyArrayList<E> implements IMyArrayList<E>{  
 private Object[] objects= new Object[16];  
 private int currentIndex=0;  
 private int size=16;  
  
 private MyIterator<E> myIterator;  
  
 /\*\*  
 \* @since 1.6  
 \* @param v  
 \* @return  
 \*/  
 public MyArrayList add(E v){  
 if(this.currentIndex==this.size){  
 System.out.println(this.currentIndex);  
 this.size\*=2;  
// this.objects= copy();  
 Object[] newObjects=new Object[this.size];  
 System.arraycopy(this.objects,0,newObjects,0,this.objects.length);  
 this.objects=newObjects;  
 }  
 System.out.println(this.currentIndex);  
 this.objects[this.currentIndex++]=v;  
 return this;  
 }  
  
 public MyIterator<E> iterator(){  
 this.myIterator=new MyArraylistIterator<E>(this);  
 return this.myIterator;  
 }  
  
 public boolean hasNext(){  
 return myIterator.hasNext();  
 }  
  
 public E next(){  
 return myIterator.next();  
 }  
  
 private Object[] copy(){  
 Object[] newObject=new Object[this.size];  
 for(int i=0;i<this.currentIndex;i++){  
 newObject[i]=this.objects[i];  
 }  
  
 return newObject;  
 }  
  
 private class MyArraylistIterator<E> implements MyIterator<E>{  
 private MyArrayList<E> myArrayList;  
 private int cursor;  
  
 public MyArraylistIterator(MyArrayList<E> myArrayList){  
 this.myArrayList=myArrayList;  
 }  
  
 public boolean hasNext() {  
 if(cursor>=this.myArrayList.getCurrentIndex())return false;  
 return true;  
 }  
  
 public E next() {  
 return (E)this.myArrayList.getObjects()[cursor++];  
 }  
 }  
  
 public Object[] getObjects() {  
 return objects;  
 }  
  
 public void setObjects(Object[] objects) {  
 this.objects = objects;  
 }  
  
 public int getCurrentIndex() {  
 return currentIndex;  
 }  
  
 public void setCurrentIndex(int currentIndex) {  
 this.currentIndex = currentIndex;  
 }  
  
 public int getSize() {  
 return size;  
 }  
  
 public void setSize(int size) {  
 this.size = size;  
 }  
}

package com.test.designpattern.\_004;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class MyArrayList2<E> implements IMyArrayList<E>{  
 private Node<E> nodeHead;  
 private int size;  
 private Node<E> lastNode;  
  
 private MyArrayList2Iterator<E> myArrayList2Iterator;  
  
 public MyArrayList2 add(E v){  
 Node node=new Node(v,null);  
 if(size==0) {  
 this.nodeHead=node;  
 this.lastNode=node;  
 }else{  
 this.lastNode.next=node;  
 this.lastNode=node;  
 }  
  
 this.size++;  
 return this;  
 }  
  
 public MyIterator<E> iterator(){  
 this.myArrayList2Iterator=new MyArrayList2Iterator<E>(this);  
 return this.myArrayList2Iterator;  
 }  
 public boolean hasNext(){  
 return this.myArrayList2Iterator.hasNext();  
 }  
 public E next(){  
 return this.myArrayList2Iterator.next();  
 }  
  
 private class Node<E>{  
 private E v;  
 private Node<E> next;  
  
 public Node(E v,Node<E> n){  
 this.v=v;  
 this.next=n;  
 }  
  
 public E getV() {  
 return v;  
 }  
  
 public void setV(E v) {  
 this.v = v;  
 }  
 }  
  
 private class MyArrayList2Iterator<E> implements MyIterator<E>{  
 private MyArrayList2<E> myArrayList2;  
 private Node<E> currentNode;  
  
 public MyArrayList2Iterator(MyArrayList2<E> myArrayList2){  
 this.myArrayList2=myArrayList2;  
 this.currentNode=myArrayList2.nodeHead;  
 }  
  
 public boolean hasNext() {  
 return this.currentNode.next!=null;  
 }  
  
 public E next() {  
 Node<E> thatNode= this.currentNode.next;  
 this.currentNode=this.currentNode.next;  
 return thatNode.getV();  
 }  
 }  
}

接口隔离原则，尽量不要暴露给不需要的对象多余的信息。