定义

数组是用来存放同一种数据类型的集合，注意只能存放同一种数据类型(Object类型数组除外)

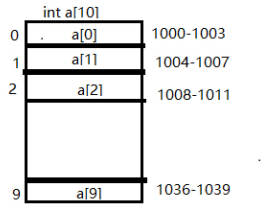
说明：在Python、Java等语言中一个数组中只能存在一种数据类型的数据，而JavaScript中的数组可以存放不同数据类型的数据。不同这种类型是非标准的数组。这里只讨论标准意义的数组

特点

1 数据的长度大小是固定的
2 数组中的元素存储在内存上连续的
3 数组是顺序线性表，其中的元素排列可以是有序的也可以是无序的

存储方式

比如一个长度为10的int类型的数组int[] a = new int[10]

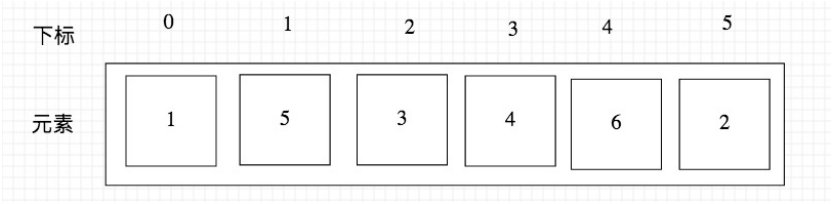


从图中可以看到，计算机给数组a分配了一块连续的内存1000-1039.其中内存块的首地址为base_address=1000

计算机会给每一个内存单元分配一个地址，计算机通过地址来访问内存中的数据。当计算机要访问数组中的某个元素的时

```
a[i]_address = base_address + i*data_type_size
```

无序数组



优点

写入快：通过下标，直接计算可获取到地址

缺点

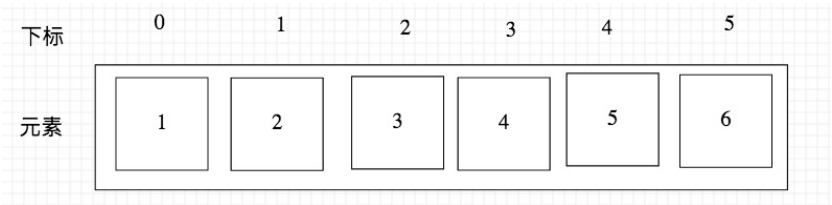
查找慢：需要遍历，最大可能遍历数组长度的次数

根据下标删除快：直接将该下标的元素清空即可

根据元素值删除慢：需要找到元素，最大可能遍历数组长度的次数

大小固定：大小是固定的，存储数据不能超过最初定义的长度

有序数组



优点

查找比无序数组要快，因为只需要遍历元素存储个数，而无序需要遍历长度为 数组长度

缺点

写入慢：需要维护元素的顺序，插入相应的地址，后面的元素还要都向后移动

删除慢：移除之后，还要移动后续元素。 可以考虑的另外方式是，进行标记，将要删除的元素状态标记为 已删除

大小固定：大小是固定的，存储数据不能超过最初定义的长度

动态数组

动态数组是数组的一种扩展，同样分为有序和无序

动态数组是为了解决数组长度大小无法扩展而实现的特殊数组，其原理是当数组的长度满了，那么就重新申请一个新的数组，长度是现到新的数组中，然后引用指向新的数组，从而达到动态扩容的效果

动态数组除了能够解决动态扩容外，其他数组的特性是完全一样的

无序动态数组实现

```
package day1.array;

/**
 * 动态数组 - 无序
 * @param <E>
```

```
 */
public class MyArray<E> {

    private int eleCount;                // 元素存放的个数
     private E[] data;                    // 数组元素
     private static int initSize = 10;    // 数组初始化长度


     /**
      * 数组初始化
      * @param initSize 数组长度
      */
     public MyArray(int initSize){
         this.initSize = initSize;
         data = (E[]) new Object[initSize];
         this.eleCount = 0;
     }

     /**
      * 数组初始化
      */
     public MyArray(){
         this(initSize);
     }

     /**
      * 获取数组长度
      * @return
      */
     public int lenth(){
         return initSize;
     }

     /**
      * 获取数组中元素的个数
      * @return
      */
     public int size(){
         return eleCount;
     }

     /**
      * 判断是否为空
      */
     public boolean isEmpty(){
         return eleCount == 0 ? true : false;
     }

     /**
      * 判断是否已满
      */
     public boolean isFull(){
         return eleCount == initSize ? true : false;
     }
```

```
60
61      /**
62       * 添加元素 : 遍历数组，遇到 null ，则加入
63       * @param e
64       * @throws Exception
65       */
66      public void add(E e) throws Exception {
67          if (isFull()){
68              E[] newDate = (E[]) new Object[initSize * 2];
69              arrayCopy(data, newDate);
70              initSize = initSize * 2;
71              data = newDate;
72              System.out.println("扩容");
73              data[eleCount] = e;
74              eleCount++;
75          }else {
76              int _index = -1;
77              for (int i = 0; i < initSize; i++){
78                  E _e = data[i];
79                  if (_e == null){
80                      _index = i;
81                      break;
82                  }
83              }
84              if (_index > -1){
85                  data[_index] = e;
86                  eleCount++;
87              }
88          }
89      }
90
91      /**
92       * 添加元素到指定位置
93       * @param e
94       * @param index
95       * @throws Exception
96       */
97      public void add(E e, int index) throws Exception {
98          if (index < 0 || index >= initSize){
99              throw new Exception("数组下标越界");
100          }
101          data[index] = e;
102      }
103
104      /**
105       * 获取指定元素在数组中的位置
106       * @param e
107       * @return
108       */
109      public int find(E e){
110          if (e == null){
111              throw new NullPointerException("查找元素不能为空");
112          }
113          int i = 0;
```

```
        while (i < initSize){
            if (e.equals(data[i])){
                return i;
            }
            i++;
        }
        return -1;
    }

    /**
     * 根据下标移除元素
     * @param index
     */
    public boolean remove(int index) throws Exception {
        if (index < 0 || index >= initSize){
            throw new Exception("数组下标越界");
        }
        if (data[index] !=null){
            data[index] = null;
            eleCount--;
            return true;
        }
        return false;
    }

    public boolean remove(E e){
        int index = find(e);
        if (index > -1){
            data[index] = null;
            eleCount--;
            return true;
        }
        return false;
    }

    public void arrayCopy(E[] src, E[] dest){
        for (int i = 0 ; i < src.length ; i++){
            dest[i] = src[i];
        }
    }

    /**
     * 根据下标获取元素
     * @param index
     * @return
     */
    public E get(int index) throws Exception {
        if (index < 0 || index >= initSize){
            throw new Exception("数组下标越界");
        }
        return data[index];
    }

    @Override
```

```
168     public String toString() {
169         StringBuffer buffer = new StringBuffer();
170         int index = 0;
171         for (E e : data){
172             if (e == null){
173                 buffer.append("null");
174             }else {
175                 buffer.append(e.toString());
176             }
177             if (index < initSize - 1){
178                 buffer.append(",");
179             }
180             index++;
181         }
182         return "数组信息 >>  长度 :  " + lenth() + " , " + "元素数量" + size() + " , 数组元素 :  " +  "
183     }
184 
185     public void printMsg(){
186         System.out.println(toString());
187     }
188 
189     public static void main(String[] args) throws Exception {
190 
191         System.out.println("初始化长度 3");
192         MyArray<String> myArray = new MyArray<>(3);
193 
194         myArray.printMsg();
195 
196         System.out.println("增加 a ");
197         myArray.add("a");
198         System.out.println("增加 b ");
199         myArray.add("b");
200 
201         myArray.printMsg();
202 
203         System.out.println("遍历开始");
204         for (int i = 0; i < myArray.lenth(); i++){
205             String s = myArray.get(i);
206             System.out.println(s);
207         }
208         System.out.println("遍历结束");
209 
210         System.out.println("写入 c ");
211         myArray.add("c");
212         System.out.println("写入 d ");
213         myArray.add("d");
214         myArray.printMsg();
215 
216         int index = myArray.find("c");
217         if (index == -1){
218             System.out.println("在数组中没有找到 c ");
219         }else {
220             System.out.println("查找 c 的在数组中的第 " + (index + 1) + " 个");
221         }
```

```
222
223         index = myArray.find("cc");
224         if (index == -1){
225             System.out.println("在数组中没有找到 cc ");
226         }else {
227             System.out.println("查找 cc 的在数组中的第 " + (index + 1) + " 个");
228         }
229
230         System.out.println("移除下标为1的元素");
231         boolean b = myArray.remove(1);
232         if (b){
233             System.out.println("移除成功");
234         }else {
235             System.out.println("移除失败");
236         }
237         myArray.printMsg();
238
239         System.out.println("移除元素 d ");
240         b = myArray.remove("d");
241         if (b){
242             System.out.println("移除成功");
243         }else {
244             System.out.println("移除失败");
245         }
246         myArray.printMsg();
247
248         System.out.println("增加 1 ");
249         myArray.add("1");
250         System.out.println("增加 2 ");
251         myArray.add("2");
252         System.out.println("增加 3 ");
253         myArray.add("3");
254         System.out.println("增加 4 ");
255         myArray.add("4");
256
257         myArray.printMsg();
258
259         System.out.println("增加 x");
260         myArray.add("x");
261         myArray.printMsg();
262     }
263 }
264
265 /* 输出
266
267 初始化长度 3
268 数组信息 >>  长度 :  3 , 元素数量0 , 数组元素 :  [null,null,null]
269 增加 a
270 增加 b
271 数组信息 >>  长度 :  3 , 元素数量2 , 数组元素 :  [a,b,null]
272 遍历开始
273 a
274 b
275 null
```

```
遍历结束
写入 c
写入 d
扩容
数组信息 >>  长度 ： 6 ， 元素数量4 ， 数组元素 ： [a,b,c,d,null,null]
查找 c 的在数组中的第 3 个
在数组中没有找到 cc
移除下标为1的元素
移除成功
数组信息 >>  长度 ： 6 ， 元素数量3 ， 数组元素 ： [a,null,c,d,null,null]
移除元素 d
移除成功
数组信息 >>  长度 ： 6 ， 元素数量2 ， 数组元素 ： [a,null,c,null,null,null]
增加 1
增加 2
增加 3
增加 4
数组信息 >>  长度 ： 6 ， 元素数量6 ， 数组元素 ： [a,1,c,2,3,4]
增加 x
扩容
数组信息 >>  长度 ： 12 ， 元素数量7 ， 数组元素 ： [a,1,c,2,3,4,x,null,null,null,null,null]

*/
```

有序动态数组实现

```java
package day1.array;

import javax.sound.midi.Soundbank;
import java.util.Comparator;

public class MyArraySort<E>{

    private int eleCount;                // 元素存放的个数
    private E[] data;                    // 数组元素
     private static int initSize = 10;    // 数组初始化长度
     private Comparator<E> comparator;

     /**
      * 数组初始化
      * @param initSize 数组长度
      * @param comparator 比较方式
      */
     public MyArraySort(int initSize, Comparator<E> comparator){
         this.initSize = initSize;
         data = (E[]) new Object[initSize];
         this.eleCount = 0;
         this.comparator = comparator;
     }

     /**
      * 数组初始化
      * @param comparator 比较方式
      */
```

```
29      public MyArraySort(Comparator<E> comparator){
30          this(initSize, comparator);
31      }
32
33      /**
34       * 数组初始化
35       * @param initSize 数组长度
36       */
37      public MyArraySort(int initSize){
38          this(initSize, new Comparator<E>() {
39              @Override
40              public int compare(E o1, E o2) {
41                  if (o1.hashCode() > o2.hashCode()){
42                      return 1;
43                  }else if (o1.hashCode() < o2.hashCode()){
44                      return -1;
45                  }
46                  return 0;
47              }
48          });
49      }
50
51      /**
52       * 数组初始化
53       */
54      public MyArraySort(){
55          this(initSize, new Comparator<E>() {
56              @Override
57              public int compare(E o1, E o2) {
58                  if (o1.hashCode() > o2.hashCode()){
59                      return 1;
60                  }else if (o1.hashCode() < o2.hashCode()){
61                      return -1;
62                  }
63                  return 0;
64              }
65          });
66      }
67
68      /**
69       * 获取数组长度
70       * @return
71       */
72      public int lenth(){
73          return initSize;
74      }
75
76      /**
77       * 获取数组中元素的个数
78       * @return
79       */
80      public int size(){
81          return eleCount;
82      }
```

```
83
84      /**
85       * 判断是否为空
86       */
87      public boolean isEmpty(){
88          return eleCount == 0 ? true : false;
89      }
90
91      /**
92       * 判断是否已满
93       */
94      public boolean isFull(){
95          return eleCount == initSize ? true : false;
96      }
97
98      /**
99       * 根据下标获取元素
100       * @param index
101       * @return
102       * @throws Exception
103       */
104      public E get(int index) throws Exception {
105          if (index < 0 || index >= initSize){
106              throw new Exception("数组下标越界");
107          }
108          return data[index];
109      }
110
111      /**
112       * 获取元素的下标，找不到返回 -1
113       * @param e
114       * @return
115       */
116      public int find(E e){
117          if (e == null){
118              throw new NullPointerException("查找元素不能为空");
119          }
120          int i = 0;
121          while (i < eleCount){
122              if (e.equals(data[i])){
123                  return i;
124              }
125              i++;
126          }
127          return -1;
128      }
129
130      /**
131       * 添加元素
132       * @param e
133       */
134      public void add(E e){
135          if (e == null){
136              throw new NullPointerException("元素不能为空");
```

```
        }
        if (isFull()){
            System.out.println("扩容");
            E[] newDate = (E[]) new Object[initSize * 2];
            int _index = _getComparatorIndex(e);
            if (_index == -1){  // 追加到末尾
                arrayCopy(data, newDate, 0, 0, eleCount);
                newDate[eleCount] = e;
            }else {
                arrayCopy(data, newDate, 0, 0, _index);
                arrayCopy(data, newDate, _index, _index + 1, eleCount - _index);
                newDate[_index] = e;
            }
            initSize = initSize * 2;
            data = newDate;
            eleCount++;
        }else {
            _add(e);
        }
    }

    public void remove(int index){
        if (index < 0 || index >= eleCount){
            throw new ArrayIndexOutOfBoundsException("数组下标越界");
        }
        if (index == eleCount - 1){
            data[index] = null;
            eleCount--;
        }else {
            for (int i = index ; i < eleCount - 1 ; i++){
                data[i] = data[i + 1];
            }
            eleCount--;
        }
    }

    public boolean remove(E e){
        int index = find(e);
        if (index == -1){
            return false;
        }
        remove(index);
        return true;
    }

    private void arrayCopy(E[] src, E[] dest, int srcOffset, int destOffset, int len) {
        int j = 0;
        for (int i = 0; i < len; i++){
            E e = src[srcOffset + j];
            dest[destOffset + j] = e;
            j++;
        }
    }

```

```
    private int _getComparatorIndex(E e){
        int _index = -1;
        for (int i = 0; i < eleCount; i++){
            int cmp = this.comparator.compare(e, data[i]);
            if (cmp < 0){
                _index = i;
                break;
            }
        }
        return _index;
    }

    private void _add(E e) {
        int _index = _getComparatorIndex(e);
        if (_index == -1){// 追加到末尾
            data[eleCount] = e;
        }else {
            for (int i = eleCount; i > _index; i--){
                data[i] = data[i - 1];
                if (i - 1 > 0){
                    data[i - 1] = data[i - 2];
                }
            }
            data[_index] = e;
        }
        eleCount++;
    }


    @Override
    public String toString() {
        StringBuffer buffer = new StringBuffer();
        for (int i = 0; i < eleCount; i++){
            buffer.append(data[i].toString());
            if (i < eleCount - 1){
                buffer.append(",");
            }
        }
        return "数组信息 >>  长度 :  " + lenth() + " , " + "元素数量" + size() + " , 数组元素 :  " +  "
    }

    public void printMsg(){
        System.out.println(toString());
    }

    public static void main(String[] args) {

        System.out.println("初始化长度为 " + 3);
        /*MyArraySort<String> myArraySort = new MyArraySort<>(5, new Comparator<String>() {
            @Override
            public int compare(String o1, String o2) {
                if (o1.length() < o2.length()){
                    return 1;
                }else if (o1.length() > o2.length()){
```

```
                    return -1;
                }
                return 0;
            }
        });*/

        MyArraySort<String> myArraySort = new MyArraySort<>(3);

        System.out.println("增加 d ");
        myArraySort.add("d");
        System.out.println("增加 c ");
        myArraySort.add("c");
        System.out.println("增加 b ");
        myArraySort.add("b");
        System.out.println("增加 a ");
        myArraySort.add("a");
        System.out.println("增加 e ");
        myArraySort.add("e");

        myArraySort.printMsg();


        System.out.println("查找 a 的下标为: " + myArraySort.find("a"));
        System.out.println("查找 x 的下标为: " + myArraySort.find("x"));

        System.out.println("移除下标 2 ");
        myArraySort.remove(2);
        myArraySort.printMsg();
    }
}


/*输出
初始化长度为 3
增加 d
增加 c
增加 b
增加 a
扩容
增加 e
数组信息 >>  长度 :  6 , 元素数量5 , 数组元素 :  [a,b,c,d,e]
查找 a 的下标为: 0
查找 x 的下标为: -1
移除下标 2
数组信息 >>  长度 :  6 , 元素数量4 , 数组元素 :  [a,b,d,e]
*/
```