



# Chapter 5 Java Collection



[javacose@qq.com](mailto:javacose@qq.com)

**Xiang Zhang**



# Content

2

- Arrays
- Collection
  - ArrayList
  - LinkedList
- Map
  - HashMap
- Iterator



# Array

3

- Declaration and Assignment of an Array:

```
int[] a = new int[10];  
for(int i=0; i<a.length; i++){  
    a[i] = i*2;  
}  
...  
String[] b = {"Hello", "World!"};  
  
int[] b = new int[2]{1,5};  
int[] b = new int[]{1,5};
```



# Array Operation

4

- Traverse (遍历)
- Min / Max
- Sum / Average / Length
- Search
- Sort
- Equal Judgment



# Array

5

- `java.util.Arrays` provides some static methods

- Binary Search

```
String[] str = {"Kobe", "Tmac", "Lebron"};  
Arrays.sort(str);  
System.out.println(Arrays.binarySearch(str, "Tmac"));
```

- Quick Sort

```
String[] str = {"Kobe", "Tmac", "Lebron"};  
Arrays.sort(str);  
System.out.println(Arrays.toString(str));
```



# Think

6

- How to write a case-insensitive `binarySearch`?
- OR, how to write a case-insensitive quick sort?

```
String[] str = {"Kobe", "Tmac", "Lebron"};  
Arrays.sort(str);  
System.out.println(Arrays.binarySearch(str, "kobe"));
```

- How to get all hit elements in binary search?



# Array

7

- copyOf / copyOfRange

```
String[] str = {"Kobe", "Tmac", "Lebron"};  
String[] anotherStr = Arrays.copyOfRange(str, 0, 1);  
System.out.println(Arrays.toString(anotherStr));
```

- fill

```
String[] str = new String[10];  
Arrays.fill(str, "Kobe");  
System.out.println(Arrays.toString(str));
```



# Array

8

- toString
- hashCode
- equals

```
String[] str1 = new String[10];  
String[] str2 = new String[10];  
Arrays.fill(str1, "Kobe");  
Arrays.fill(str2, "Kobe");  
System.out.println(Arrays.hashCode(str1));  
System.out.println(Arrays.hashCode(str2));  
System.out.println(Arrays.equals(str1, str2));  
System.out.println(Arrays.toString(str1));
```

```
<terminated> ArrayTest [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (2009-10-  
103841569  
103841569  
true  
[Kobe, Kobe, Kobe, Kobe, Kobe, Kobe, Kobe, Kobe, Kobe, Kobe]
```





# About Hash Algorithm

9

哈希算法将任意长度的二进制值映射为较短的固定长度的二进制值，这个小的二进制值称为哈希值。哈希值是一段数据唯一且极其紧凑的数值表示形式。如果散列一段明文而且哪怕只更改该段落的一个字母，随后的哈希都将产生不同的值。要找到散列为同一个值的两个不同的输入，在计算上是不可能的，所以数据的哈希值可以检验数据的完整性。一般用于快速查找和加密算法。

来自百度百科 <http://baike.baidu.com/view/273836.htm>



# Array

10

- Shortage of Array
  - Fixed-length
  - Complex for insert and delete
  - ??? How to insert and delete?



# Collections

11

- Variable Length
- A Relation Between Key and Value
- More ways to visit values
- Java Collection Framework in `java.util`



# 集合框架

12

- `Collection<E>` (集合)
  - `Set<E>` // non-repeat
    - ✧ `SortedSet<E>` // sorted
  - `List<E>` // sequential, repeatable
  - `Queue<E>` // FIFO
- `Map<K,V>` (映射)
  - ✧ `HashMap<K,V>` //unsorted
  - ✧ `SortedMap<K,V>` // sorted
- `Iterator<E>` (迭代器)



# Collection - ArrayList

13

- Sequential and Linear (线性表)
- Use Array as Backend (动态数组)
- Variable Length
- Methods:
  - add(Object element) / remove(Object element)
  - add(int index, Object element) / remove(int index)
  - get(int index) / set(int index)
  - indexOf(Object o)
  - clear() / isEmpty()
  - Size() / toArray()



# Attention

14

- Get the number of elements:

- For ArrayList

```
ArrayList list = new ArrayList();  
list.add("Kobe");  
System.out.println(list.size());
```

- For Array

```
String[] list = new String[10];  
System.out.println(list.length);
```



# List - ArrayList

15

- ArrayList Creation // notice the capacity of ArrayList

```
ArrayList<String> list = new ArrayList<String>() //初始化时initial capacity为10;  
ArrayList<String> list = new ArrayList<String>(100) //指定initial capacity;  
list.ensureCapacity(10000) //修改capacity
```



# List - ArrayList

16

- Methods :

```
list.add("Kobe"); //增加
list.add("Tmac"); //增加
list.remove("Tmac"); //删除
list.remove(0); //删除
list.clear(); //清空
list.add("Lebron"); //增加
list.contains("Lebron"); //是否包含某元素? true
list.get(0); //访问
list.set(0, "Kobe"); //修改
list.indexOf("Kobe"); //查找
list.isEmpty(); //false
Iterator<String> it = list.iterator(); //返回迭代器
```





# List - ArrayList

17

- Feature
  - Efficient in random access of elements
  - May enlarge backend array when append new elements (can be partly solved by setting initial capacity)
  - Not efficient for insertion (may cause the movement of elements)
  - Waste of space (solved by trimToSize)



# List - LinkedList

18

- Implemented by co-reference of neighbors
- No capacity
- Each Element stores:
  - A reference to the previous element
  - A reference to the succeeded element
  - The value





# List - LinkedList

19

- Methods

```
list.add("Kobe"); //增加  
list.addFirst("Tmac"); //在首部增加  
list.addLast("Lebron"); //在尾部增加  
list.addLast("Paul"); //在尾部增加  
list.removeFirst(); //在首部删除  
list.removeLast(); //在尾部删除  
list.add(1, "Tmac"); //插入
```



# List - LinkedList

20

- Feature
  - Do not cause the reassignment of memory
  - Efficient for add / delete / insert
  - Not efficient for random access (need traverse from head)



# Performance Benchmark

21

- ArrayList <String> vs. LinkedList<String>
- Both List have 1000 elements
- 10000 runs
  - get(i), where i from 0-1000
  - Traverse all the elements: iteration
  - Insert 100000 elements in the middle
  - Delete one by one

```
LinkedList<String> list = new LinkedList<String>();  
list.add("kobe");  
list.add("Tmac");  
list.add("Lebron");  
Iterator<String> iterator = list.iterator();  
while(iterator.hasNext()){  
    System.out.println(iterator.next());  
}
```



# Performance Benchmark

22

	ArrayList	LinkedList
get (ms)	172	3297
iteration (ms)	813	328
insert (ms)	140	16
remove (ms)	4625	15



# Map - HashMap

23

- HashMap - a Mapping Between Key and Value
- Example: Student ID - Name

学号	姓名
71108501	Tom
71108502	Mike
71108503	Peter
...	...

- Feature: Search a value by key efficiently



# Map - HashMap

24

- Methods

```
HashMap<Integer, String> map = new HashMap<Integer, String>();  
map.put(71108501, "Tom"); //增加  
map.put(71108502, "Mike"); //增加  
map.put(71108503, "Peter"); //增加  
String name = map.get(71108502); //查找  
map.remove(71108503); //删除  
Set keySet = map.keySet(); //获取所有的key  
Collection valueSet = map.values(); //获取所有的value  
Set entrySet = map.entrySet(); //获取所有的entry
```

?? Why use Set as the return type??







# Iterator

25

- Iterator for the Traverse of Collection
- There is an iterator() Method in Collection
  - Each implemented class of Collection should implemented iterator()
  - Each implemented class of Collection can be traversed using iterator()
- Methods in Iterator:
  - hasNext()
  - next()



# Example for LinkedList

26

```
LinkedList<String> list = new LinkedList<String>();  
list.add("kobe");  
list.add("Tmac");  
list.add("Lebron");  
Iterator<String> iterator = list.iterator();  
while(iterator.hasNext()){  
    System.out.println(iterator.next());  
}
```



# Example of HashMap

27

- Using Iterator to traverse HashMap

```
HashMap<Integer, String> map = new HashMap<Integer, String>();  
map.put(71108501, "Tom"); //增加  
map.put(71108502, "Mike"); //增加  
map.put(71108503, "Peter"); //增加  
/* 遍历 */  
Iterator<Integer> it = map.keySet().iterator();  
while(it.hasNext()){  
    Integer key = it.next();  
    String value = map.get(key);  
    System.out.println("Key:" + key + " value:" + value);  
}
```



# For-each loop

28

- For Loop :
  - for(int i=0; i<10; i++)
- For-each Loop
  - ArrayList list = new...
  - for(int i: list)
  - For-each loop means "for each element in a collection ..."

```
int[] array = {1,2,3};  
for(int i=0; i<array.length; i++){  
    System.out.println(array[i]);  
}
```

```
int[] array = {1,2,3};  
for(int i:array){  
    System.out.println(i);  
}
```

```
Collection<Integer> col;  
// col 初始化  
for(int i:col){  
    System.out.println(i);  
}
```



# Self-study

29

- Java Class Library

- java.lang – Java Language Related
- java.io – Input and Output
- java.math – Mathematical Calculation
- java.net – Network Programming
- java.nio – New I/O
- java.text – Text Processing
- java.util – Useful Toolkit ( Collection, Date, Time, etc. )



# Self-study

30

- Java Utility
  - Formatter //create formatted text
  - Observer/Observable //Observer design principle
  - Math and Random //generation of random numbers
  - Timer/TimerTask //Timer and Scheduler
- Readings: Chapter 22



# Forecast

31

- Significance of Generic Type
- Definition of Generic Type
- Usage of Generic Type