

Стандартная библиотека Java: Collections Framework

Алексей Владыкин

21 октября 2013

1 Collections Framework

2 Generics



Что такое коллекции

- Разнообразные контейнеры для хранения наборов объектов, более удобные, чем массивы

Что такое коллекции

- Разнообразные контейнеры для хранения наборов объектов, более удобные, чем массивы
 - добавление/удаление элементов
 - запрет модификации коллекции
 - операции поиска, объединения, вычитания
 - индексация элементов произвольными объектами
- В отличие от массивов, могут хранить только объекты. Для примитивных типов — классы-обертки и autoboxing
- Пакет `java.util`

java.util.Collection

- Базовый интерфейс для коллекций
- Основные операции:
 - `int` `size()`
 - `boolean` `isEmpty()`
 - `boolean` `contains(Object o)` — использует `equals`
 - `boolean` `add(E e)`
 - `boolean` `remove(Object o)`
 - `void` `clear()`

java.util.Iterator

- Единообразный способ обхода элементов коллекции
- Операции:
 - `boolean hasNext()`
 - `E next()`
 - `void remove()`

Использование итератора

```
// foreach syntax
for (E element : collection) {
    System.out.println(element);
}

// equivalent to
Iterator<E> it = collection.iterator();
while (it.hasNext()) {
    E element = it.next();
    System.out.println(element);
}
```

Разновидности коллекций

- List — список
(фиксированный порядок, доступ к элементам по индексу)
- Queue, Deque — очередь и дек
(доступ к элементам с начала и с конца)
- Set — множество
(каждый элемент встречается не более одного раза)
- Map — ассоциативный массив
(набор пар «ключ–значение»)

java.util.List

- Фиксированный порядок
- Доступ к элементам по индексу
- Операции:
 - E get(int index)
 - E set(int index, E element)
 - void add(int index, E element)
 - E remove(int index)
 - int indexOf(Object o)
 - int lastIndexOf(Object o)
 - List<E> subList(int fromIndex, int toIndex)
- equals: списки равны, если содержат равные элементы в одинаковом порядке

java.util.ArrayList

- Реализация списка на основе массива
- Специфические операции:
 - `void ensureCapacity(int capacity)`
 - `void trimToSize()`
- Эффективный доступ к элементу по индексу
- Вставка/удаление по индексу имеет линейную трудоемкость

```
List<String> words = new ArrayList<>();  
words.add("one");  
words.set(0, "two");  
words.add(0, "three");  
words.remove(1);
```

java.util.LinkedList

- Реализация списка на основе двусвязного списка
- Эффективные вставка и удаление элемента в начале и в конце списка
- Доступ к элементу по индексу имеет линейную трудоемкость

```
List<String> words = new LinkedList<>();  
words.add("one");  
words.add("two");  
words.add("three");  
words.subList(1, 3).clear();
```

java.util.Queue

- First In — First Out (FIFO)
- Операции:
 - `boolean offer(E e)`
 - `E peek()`
 - `E poll()`

java.util.PriorityQueue

- Реализация очереди с приоритетами на основе двоичной кучи
- offer и poll работают за $O(\log(N))$
- Извлекается элемент с минимальным значением

```
PriorityQueue<Integer> pq = new PriorityQueue<>();
pq.offer(3);
pq.offer(2);
pq.offer(1);

Integer element;
while ((element = pq.poll()) != null) {
    System.out.println(element);
}
```

java.util.Deque

- Стек и очередь в одном флаконе
- Операции:
 - `boolean offerFirst(E e)`
 - `E peekFirst()`
 - `E pollFirst()`
 - `boolean offerLast(E e)`
 - `E peekLast()`
 - `E pollLast()`

java.util.ArrayDeque

- Реализация дека на основе массива
- Рекомендованный класс для обычной очереди и стека

```
Deque<Integer> pq = new ArrayDeque<>();  
pq.offerLast(3);  
pq.offerLast(2);  
pq.offerLast(1);  
  
Integer element;  
while ((element = pq.pollFirst()) != null) {  
    System.out.println(element);  
}
```

java.util.Set

- Каждый элемент встречается не более одного раза
- Не добавляет новых операций к тем, что есть в `java.util.Collection`
- Но гарантирует, что при добавлении элементов дубликаты не появятся
- `equals`: множества равны, если содержат одинаковые элементы

java.util.HashSet

- Реализация множества на основе хеш-таблицы
- Порядок обхода элементов непредсказуем

```
Set<String> words = new HashSet<>();  
words.add("one");  
words.add("one");  
words.add("two");  
words.add("two");
```

java.util.LinkedHashSet

- Реализация множества на основе хеш-таблицы
- Порядок обхода элементов определяется порядком вставки

```
Set<String> words = new LinkedHashSet<>();  
words.add("one");  
words.add("one");  
words.add("two");  
words.add("two");
```

Специфика хеш-таблиц

- Контракт `equals()` и `hashCode()`:
если `a.equals(b)`, то `a.hashCode()==b.hashCode()`
- Пока объект находится в хеш-таблице, нельзя менять его поля, влияющие на значение `hashCode()`

java.util.TreeSet

- Реализация множества на основе дерева поиска
- Элементы хранятся отсортированными

```
SortedSet<String> words = new TreeSet<>();  
words.add("aaa");  
words.add("bbb");  
words.add("ccc");  
words.headSet("bbb").clear();
```

Специфика деревьев поиска

- Порядок элементов определяется:
 - объектом типа `java.util.Comparator` с методом `int compare(T o1, T o2)`
 - методом элементов `int compareTo(T o)` (элементы должны реализовать интерфейс `java.lang.Comparable`)
- Пока объект находится в дереве, нельзя менять его поля, влияющие на вычисление `compareTo(T)`
- Контракт `equals()` и `compareTo()`:
`a.equals(b) == (a.compareTo(b) == 0)`

Удаление дубликатов из коллекции

```
List<String> list = new ArrayList<>();  
list.add("aaa");  
list.add("aaa");  
list.add("bbb");  
list.add("aaa");  
  
Set<String> set =  
    new LinkedHashSet<>(list);  
  
List<String> listWithoutDups =  
    new ArrayList<>(set);
```

java.util.Map

- Набор пар «ключ–значение»
- Не наследует `java.util.Collection`
- Основные операции:
 - `int` `size()`
 - `boolean` `isEmpty()`
 - `V` `get(Object key)`
 - `V` `put(K key, V value)`
 - `V` `remove(Object key)`
 - `boolean` `containsKey(Object key)`
 - `boolean` `containsValue(Object value)`
 - `Set<K>` `keySet()`
 - `Collection<V>` `values()`
 - `Set<Map.Entry<K, V>>` `entrySet()`
- `equals`: равны, если содержат одинаковые пары «ключ-значение»

java.util.HashMap

- Реализация ассоциативного массива на основе хеш-таблицы
- Порядок обхода элементов непредсказуем
- Есть `java.util.LinkedHashMap`

```
Map<String, String> dictionary = new HashMap<>();  
dictionary.put("foo", "bar");  
dictionary.put("bar", "baz");  
dictionary.remove("bar");
```

java.util.TreeMap

- Реализация ассоциативного массива на основе дерева поиска
- Элементы хранятся отсортированными по ключу

```
SortedMap<String, String> dictionary =  
    new TreeMap<>();  
dictionary.put("foo", "bar");  
dictionary.put("bar", "baz");  
dictionary.subMap("bar", "foo").clear();
```

Обход ассоциативного массива

```
Map<A, B> map = new HashMap<>();  
  
for (A key : map.keySet()) { ... }  
  
for (B value : map.values()) { ... }  
  
for (Map.Entry<A, B> entry : map.entrySet()) {  
    entry.getKey();  
    entry.getValue();  
}
```

Устаревшие классы

- `java.util.Vector`
- `java.util.Stack`
- `java.util.Dictionary`
- `java.util.Hashtable`



Сборник рецептов

Как отсортировать список

```
// move elements randomly  
Collections.shuffle(list);  
  
// list is sorted in-place  
Collections.sort(list);
```

Как запретить изменение

```
Set<String> set =  
    Collections.unmodifiableSet(originalSet);  
  
set.remove("abc");  
    // throws UnsupportedOperationException
```

Как перелить коллекцию в массив

```
// List<Integer> list

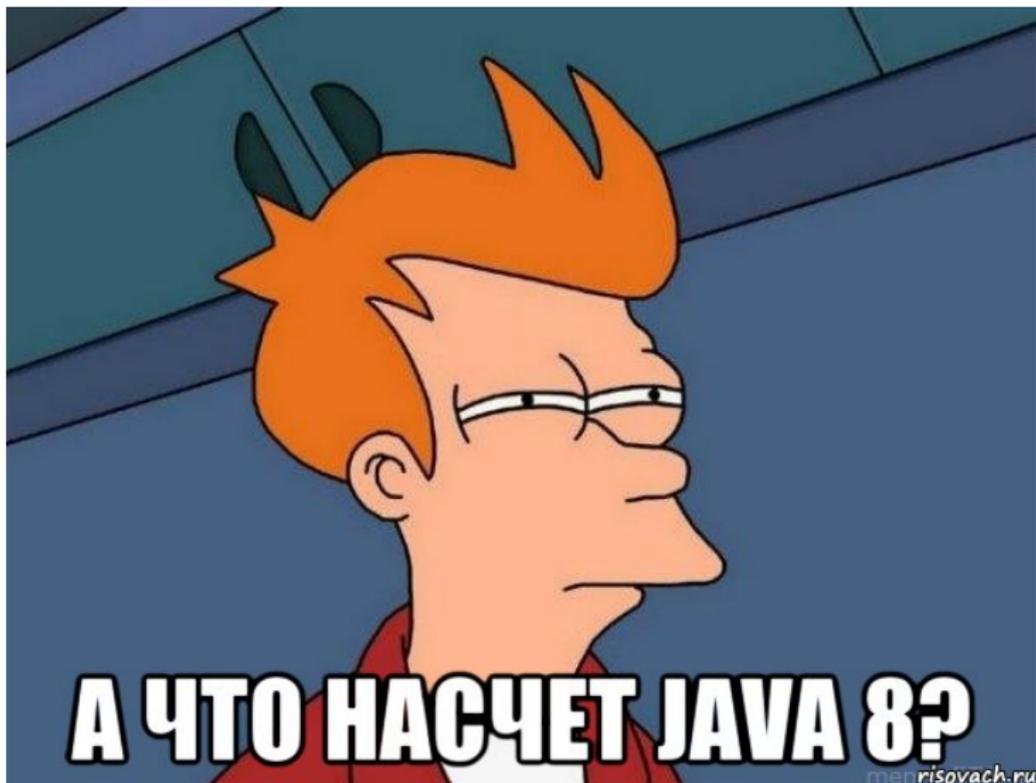
Integer[] array =
    list.toArray(new Integer[list.size()]);
```

Как перелить массив в коллекцию

```
String[] array = {"A", "B", "C"};

Set<String> set1 =
    new HashSet<>(Arrays.asList(array));

Set<String> set2 = new HashSet<>();
Collections.addAll(set2, array);
```



Было

```
List<Block> blocks = /* ... */;

int sumOfWeights = 0;
for (Block block : blocks) {
    if (block.getColor() == Color.RED) {
        sumOfWeights += block.getWeight();
    }
}
```

Стало

```
List<Block> blocks = /* ... */;  
int sumOfWeights = blocks.stream()  
    .filter(b -> b.getColor() == Color.RED)  
    .map(b -> b.getWeight())  
    .sum();
```



- Возможность параметризовать класс или метод другим классом
- Появилась в Java 5
- Напоминает шаблоны в C++, но есть большие отличия
- Нельзя использовать в качестве параметра примитивные типы или значения примитивных типов

Параметризованный класс

```
package java.lang.ref;

public abstract class Reference<T> {

    private T referent;

    public T get() {
        return referent;
    }

    /* the rest was skipped */
}
```

Параметризованный метод

```
package java.util;

public class Collections {

    public static <T> List<T> nCopies(int n, T o)
    // implementation skipped

}
```

Ограничения

- По имени параметра нельзя создать экземпляр или массив

```
T obj = new T();    // compilation error  
T[] arr = new T[]; // compilation error
```

- Во время исполнения информация о generic-параметрах недоступна

```
if (list instanceof List<T>)  
    // compilation error
```

Поддержка наследования

- Если `class Child extends Parent`, то:

```
Parent parent = new Child();           // OK
Parent[] parentArray = new Child[];    // OK

List<Parent> list =
    new ArrayList<Child>();             // not OK!

List<? extends Parent> list2 =
    new ArrayList<Child>();             // OK
```

```
package java.util;

public class Collections {

    public static <T> void copy(
        List<? super T> dest,
        List<? extends T> src)
        // implementation skipped

}
```

Autoboxing

- Автоматическая упаковка примитивных типов в обертки и распаковка обратно

```
List<Integer> list = new ArrayList<>();
for (int i = 0; i < 10; ++i) {
    list.add(i);
}
for (int i = 0; i < 10; ++i) {
    list.remove(i); // ooops
}
```

Что сегодня узнали

- Можно хранить наборы объектов не только в массивах, но и в более гибких и функциональных *коллекциях*
- В стандартной библиотеке Java есть списки, множества и ассоциативные массивы
- Работа с коллекциями стала намного удобнее с появлением Generic'ов в Java 5