Стандартная библиотека Java: пакет `java.util`

Алексей Владыкин

24 октября 2012

1. Collections Framework
2. Generics
3. Другие классы `java.util`

Что такое коллекции

- Разнообразные контейнеры для хранения наборов объектов
- Предоставляют значительно больше возможностей, чем массивы
- В первую очередь, возможность добавления и удаления элементов с динамическим изменением размера коллекции
- В отличие от массивов, могут хранить только объекты, но не примитивные типы (однако можно использовать классы-обертки)

Разновидности коллекций

- `java.util.List` — список
  (фиксированный порядок, доступ к элементам по индексу)
- `java.util.Set` — множество
  (каждый элемент встречается не более одного раза)
- `java.util.Map` — ассоциативный массив
  (набор пар «ключ–значение»)

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()`
- По возможности следует использовать цикл `foreach` вместо явной работы с итератором

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)`

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();
```

Сравнение коллекций

- Метод `boolean equals(Object obj)`
- Списки равны, если содержат равные элементы в одинаковом порядке
- Множества равны, если содержат одинаковые элементы
- Ассоциативные массивы равны, если содержат одинаковые пары «ключ-значение»

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

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 значение его полей, влияющих 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`)
- 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()`

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();
}
```

java.util.Collections

- Методы для сортировки, поиска, определения минимума и максимума
- Защита коллекций от изменения
- Экземпляры пустых коллекций

Устаревшие классы

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

- Возможность параметризовать класс или метод некоторым типом
- 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

```
public class GenericClass<T> {

    private T ref;

    public T getRef() {
        return ref;
    }

    public void setRef(T ref) {
        this.ref = ref;
    }
}
```

Параметризованный метод

```
public class ClassWithGenericMethod {

    public static <T extends Comparable<T>>
            T min(T a, T b) {
        return a.compareTo(b) <= 0 ? a : b;
    }

    public static <T extends Comparable<T>>
            T max(T a, T b) {
        return 0 <= a.compareTo(b) ? a : b;
    }
}
```

Ограничения

- По имени параметра нельзя создать экземпляр или массив: `new T(); new T[];`
- 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
```

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
}
```

- java.util.Arrays
- java.util.Objects
- java.util.Date
- java.util.Calendar
- java.util.Random

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