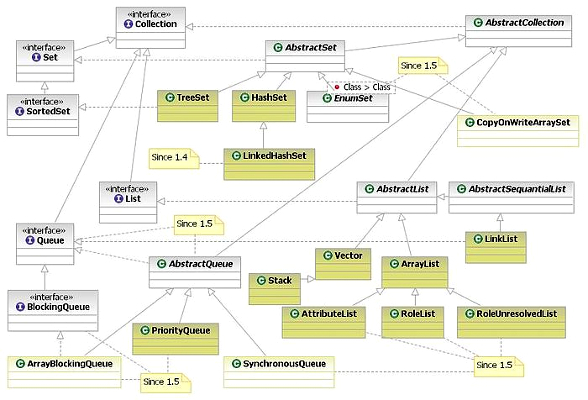
Шпаргалка Java программиста 3. Коллекции в Java (стандартные, guava, apache, trove, gs-collections и другие)

Сегодня я хотел бы поговорить о коллекциях в Java. Это тема встречается практически на любом техническом интервью Java разработчика, однако далеко не все разработчики в совершенстве освоили все коллекции даже стандартной библиотеки, не говоря уже о всех библиотеках с альтернативными реализациями коллекций, таких как guava, apache, trove и ряд других. Давайте посмотрим какие вообще коллекции можно найти в мире Java и какие методы работы с ними существуют.  
  
  
  
Эта статья полезна как для начинающих (чтобы получить общее понимание что такое коллекции и как с ними работать), так и для более опытных программистов, которые возможно найдут в ней что-то полезное или просто структурируют свои знания. Собственно, главное чтобы у вас были хотя бы базовые знания о коллекциях в любом языке программирования, так как в статье не будет объяснений что такое коллекция в принципе.

Общее оглавление 'Шпаргалок'

**I. Стандартная библиотека коллекций Java**

Естественно, практически все и так знают основные коллекции в JDK, но все-таки вспомним о них, если вы уверены, что и так все знаете о стандартных коллекциях можете смело пропускать все что в спойлерах до следующего раздела.

Замечание о коллекциях для начинающих

Иногда достаточно сложно для начинающих (особенно если они перешли из других языков программирования) понять, что в коллекции Java хранятся только ссылки/указатели и ничего более. Им кажется, что при вызове add или put объекты действительно хранятся где-то внутри коллекции, это верно только для массивов, когда они работают с примитивными типами, но никак не для коллекций, которые хранят только ссылки. Поэтому очень часто на вопросы собеседований вроде «А можно ли назвать точный размер коллекции ArrayList» начинающие начинают отвечать что-то вроде «Зависит от типа объектов что в них хранятся». Это совершенно не верно, так коллекции никогда не хранят сами объекты, а только ссылки на них. Например, можно в List добавить миллион раз один и то же объект (точнее создать миллион ссылок на один объект).

1) Интерфейсы коллекций JDK

Интерфейсы коллекций JDK

**Интерфейсы коллекций**

|  |  |
| --- | --- |
| **Название** | **Описание** |
| [Iterable](http://docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html) | Интерфейс означающий что у коллекции есть iterator и её можно обойти с помощью for(Type value:collection). Есть почти у всех коллекций (кроме Map) |
| [Collection](http://docs.oracle.com/javase/8/docs/api/java/util/Collection.html) | Основной интерфейс для большинства коллекций (кроме Map) |
| [List](http://docs.oracle.com/javase/8/docs/api/java/util/List.html) | List это упорядоченная в порядке добавления коллекция, так же известная как последовательность (sequence).  Дублирующие элементы и null в большинстве реализаций — разрешены.  Позволяет доступ поиндексу элемента.  Расширяет Collection интерфейс. |
| [Set](http://docs.oracle.com/javase/8/docs/api/java/util/Set.html) | Интерфейс реализующий работу с множествами (похожими на математические множества), дублирующие элементы запрещены.  Может быть, а может и не быть упорядоченным. Расширяет Collection интерфейс. |
| [Queue](http://docs.oracle.com/javase/8/docs/api/java/util/Queue.html) | Очередь — это коллекция, предназначенная для хранения объектов до обработки, в отличии от обычных операций над коллекциями, очередь предоставляет дополнительные методы добавление, получения и просмотра. Быстрый доступ по индексу элемента, как правило, не содержит. Расширяет Collection интерфейс |
| [Deque](http://docs.oracle.com/javase/8/docs/api/java/util/Deque.html) | Двухсторонняя очередь, поддерживает добавление и удаление элементов с обоих концов. Расширяет Queue интерфейс. |
| [Map](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html) | Работает со соответствием ключ — значение. Каждый ключ соответствует только одному значению. В отличие от других коллекций не расширяет никакие интерфейсы (в том числе Collection и Iterable) |
| [SortedSet](http://docs.oracle.com/javase/8/docs/api/java/util/SortedSet.html) | Автоматически отсортированное множество, либо в натуральном порядке (для подробностей смотрите Comparable интерфейс), либо используя Comparator. Расширяет Set интерфейс |
| [SortedMap](https://docs.oracle.com/javase/8/docs/api/java/util/SortedMap.html) | Это map'а ключи которой автоматически отсортированы, либо в натуральном порядке, либо с помощью comparator'а. Расширяет Map интерфейс. |
| [NavigableSet](http://docs.oracle.com/javase/8/docs/api/java/util/NavigableSet.html) | Это SortedSet, к которому дополнительно добавили методы для поиска ближайшего значения к заданному значению поиска. NavigableSet может быть доступен для доступа и обхода или в порядке убывания значений или в порядке возрастания. |
| [NavigableMap](https://docs.oracle.com/javase/8/docs/api/java/util/NavigableMap.html) | Это SortedMap, к которому дополнительно добавили методы для поиска ближайшего значения к заданному значению поиска. Доступен для доступа и обхода или в порядке убывания значений или в порядке возрастания. |

Интерфейсы из пакета java.util.concurrent

|  |  |
| --- | --- |
| **Название** | **Описание** |
| [BlockingQueue](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html) | Многопоточная реализация Queue, содержащая возможность задавать размер очереди, блокировки по условиям, различные методы, по-разному обрабатывающие переполнение при добавлении или отсутствие данных при получении (бросают exception, блокируют поток постоянно или на время, возвращают false и т.п.) |
| [TransferQueue](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/TransferQueue.html) | Эта многопоточная очередь может блокировать вставляющий поток, до тех пор, пока принимающий поток не вытащит элемент из очереди, таким образом с её помощью можно реализовывать синхронные и асинхронные передачи сообщений между потоками |
| [BlockingDeque](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingDeque.html) | Аналогично BlockingQueue, но для двухсторонней очереди |
| [ConcurrentMap](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentMap.html) | Интерфейс, расширяет интерфейс Map. Добавляет ряд новых атомарных методов: putIfAbsent, remove, replace, которые позволяют облегчить и сделать более безопасным многопоточное программирование. |
| [ConcurrentNavigableMap](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentNavigableMap.html) | Расширяет интерфейс NavigableMap для многопоточного варианта |

Если вам интересны более подробная информация о интерфейсах и коллекциях из java.util.concurrent советую  
прочитать вот эту [статью](http://habrahabr.ru/company/luxoft/blog/157273/). <https://habr.com/ru/company/luxoft/blog/157273/>

2) Таблица с очень кратким описанием всех коллекций

Таблица с очень кратким описанием всех коллекций

|  |  |  |
| --- | --- | --- |
| **Тип** | **Однопоточные** | **Многопоточные** |
| Lists | * ArrayList — основной список, основан на массиве * LinkedList — полезен лишь в некоторых редких случаях * Vector — устарел | * CopyOnWriteArrayList — редкие обновления, частые чтения |
| Queues /  Deques | * ArrayDeque — основная реализация, основан на массиве * Stack — устарел * PriorityQueue — отсортированная очередь | * ArrayBlockingQueue — блокирующая очередь * ConcurrentLinkedDeque / ConcurrentLinkedQueue — очередь на связанных нодах(используется алгоритм CAS) * DelayQueue — очередь с задержкой для каждого элемента * LinkedBlockingDeque / LinkedBlockingQueue — блокирующая очередь на связанных нодах * LinkedTransferQueue — может служить для передачи элементов * PriorityBlockingQueue — многопоточная PriorityQueue * SynchronousQueue — простая многопоточная очередь |
| Maps | * HashMap — основная реализация * EnumMap — enum в качестве ключей * Hashtable — устарел * IdentityHashMap — ключи сравниваются с помощью = = * LinkedHashMap — сохраняет порядок вставки * TreeMap — сортированные ключи * WeakHashMap — слабые ссылки, полезно для кешей | * ConcurrentHashMap — основная многопоточная реализация, где все данные делятся на отдельные сегменты и блокируются только отдельные сегменты при изменении * ConcurrentSkipListMap — отсортированная многопоточная реализация |
| Sets | * HashSet — основная реализация множества * EnumSet — множество из enums * BitSet\* — множество битов * LinkedHashSet — сохраняет порядок вставки * TreeSet — отсортированные set | * ConcurrentSkipListSet — отсортированный многопоточный set * CopyOnWriteArraySet — редкие обновления, частые чтения |

\* — на самом деле, BitSet хоть и называется Set'ом, интерфейс Set не наследует.

3) Устаревшие коллекции в JDK

Устаревшие коллекции Java

Универсальные коллекции общего назначения, которые признаны устаревшими (legacy)

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| [**Hashtable**](https://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html) | Изначально задумывался как синхронизированный аналог HashMap, когда ещё не было возможности получить версию коллекции используя Collecions.synchronizedMap. На данный момент как правило используют ConcurrentHashMap. HashTable более медленный и менее потокобезопасный чем синхронный HashMap, так как обеспечивает синхронность на уровне отдельных операций, а не целиком на уровне коллекции. |
| [**Vector**](https://docs.oracle.com/javase/8/docs/api/java/util/Vector.html) | Раньше использовался как синхронный вариант ArrayList, однако устарел по тем же причинам что и HashTable. |
| [**Stack**](https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html) | Раньше использовался как для построения очереди, однако поскольку построен на основе Vector, тоже считается морально устаревшим. |

Специализированные коллекции, построенные на устаревших (legacy) коллекциях

|  |  |  |
| --- | --- | --- |
| **Имя** | **Основан на** | **Описание** |
| [*Properties*](https://docs.oracle.com/javase/8/docs/api/java/util/Properties.html) | [Hashtable](https://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html) | Как структура данных, построенная на Hashtable, Properties довольно устаревшая конструкция, намного лучше использовать Map, содержащий строки. Подробнее почему Properties не рекомендуется использовать можно найти в этом [обсуждении](http://stackoverflow.com/questions/2358651/are-java-properties-effectively-deprecated). |
| [*UIDefaults*](https://docs.oracle.com/javase/8/docs/api/java/util/Properties.html) | [Hashtable](https://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html) | Коллекция, хранящая настройки по умолчанию для Swing компонент |

4) Коллекции, реализующие интерфейс **List** (список)

Коллекции, реализующие интерфейс List (список)

Универсальные коллекции общего назначения, реализующие List:

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Основан на** | **Описание** | **Размер\*** |
| [**ArrayList**](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html) | [List](https://docs.oracle.com/javase/8/docs/api/java/util/List.html) | Реализация List интерфейса на основе динамически изменяемого массива. В большинстве случаев, лучшая возможная реализация List интерфейса по потреблению памяти и производительности.  В крайне редких случаях, когда требуются частые вставки в начало или середину списка с очень малым количеством перемещений по списку, LinkedList будет выигрывать в производительности (но советую в этих случаях использовать TreeList от apache). Если интересны подробности ArrayList советую посмотреть эту [статью](http://habrahabr.ru/post/128269/). | 4\*N |
| [**LinkedList**](https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html) | [List](https://docs.oracle.com/javase/8/docs/api/java/util/List.html) | Реализация List интерфейса на основе двухстороннего связанного списка, то есть когда каждый элемент, указывает на предыдущий и следующий элемент. Как правило, требует больше памяти и хуже по производительности, чем ArrayList, имеет смысл использовать лишь в редких случаях когда часто требуется вставка/удаление в середину списка с минимальными перемещениями по списку (но советую в этих случаях использовать TreeList от apache).Так же реализует Deque интерфейс. При работе через Queue интерфейс, LinkedList действует как FIFO очередь. Если интересны подробности LinkedList советую посмотреть эту [статью](http://habrahabr.ru/post/127864/). | 24\*N |

Коллекции из пакета java.util.concurrent

|  |  |  |
| --- | --- | --- |
| **Название** | **Основан на** | **Описание** |
| [**CopyOnWriteArrayList**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CopyOnWriteArrayList.html) | [List](https://docs.oracle.com/javase/8/docs/api/java/util/List.html) | Реализация List интерфейса, аналогичная ArrayList, но при каждом изменении списка, создается новая копия всей коллекции. Это требует очень больших ресурсов при каждом изменении коллекции, однако для данного вида коллекции не требуется синхронизации, даже при изменении коллекции во время итерирования. |

Узкоспециализированные коллекции на основе List.

|  |  |  |
| --- | --- | --- |
| **Название** | **Основан на** | **Описание** |
| [*RoleList*](https://docs.oracle.com/javase/8/docs/api/javax/management/relation/RoleList.html) | [ArrayList](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html) | Коллекция для хранения списка ролей (Roles). Узкоспециализированная коллекция основанная на ArrayList с несколькими дополнительными методами |
| [*RoleUnresolvedList*](https://docs.oracle.com/javase/8/docs/api/javax/management/relation/RoleUnresolvedList.html) | [ArrayList](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html) | Коллекция для хранения списка unresolved ролей (Unresolved Roles). Узкоспециализированная коллекция основанная на ArrayList с несколькими дополнительными методами |
| [*AttributeList*](https://docs.oracle.com/javase/8/docs/api/javax/management/AttributeList.html) | [ArrayList](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html) | Коллекция для хранения атрибутов MBean. Узкоспециализированная коллекция основанная на ArrayList с несколькими дополнительными методами |

\* — размер дан в байтах для 32 битных систем и Compressed Oops, где N это capacity списка

5) Коллекции, реализующие интерфейс **Set** (множество)

Коллекции, реализующие интерфейс Set (множество)

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Основан на** | **Описание** | **Размер\*** |
| [HashSet](https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html) | [Set](https://docs.oracle.com/javase/8/docs/api/java/util/Set.html) | Реализация Set интерфейса с использованием хеш-таблиц. В большинстве случаев, лучшая возможная реализация Set интерфейса. | 32\*S + 4\*C |
| [LinkedHashSet](https://docs.oracle.com/javase/8/docs/api/java/util/LinkedHashSet.html) | [HashSet](https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html) | Реализация Set интерфейса на основе хеш-таблиц и связанного списка. Упорядоченное по добавлению множество, которое работает почти так же быстро как HashSet. В целом, практически тоже самое что HashSet, только порядок итерирования по множеству определен порядком добавления элемента во множество в первый раз. | 40 \* S + 4\*C |
| [TreeSet](https://docs.oracle.com/javase/8/docs/api/java/util/TreeSet.html) | [NavigableSet](https://docs.oracle.com/javase/8/docs/api/java/util/NavigableSet.html) | Реализация NavigableSet интерфейса, используя красно-черное дерево. Отсортировано с помощью Comparator или натурального порядка, то есть обход/итерирование по множеству будет происходить в зависимости от правила сортировки. Основано на TreeMap, так же как HashSet основан на HashMap | 40 \* S |
| [EnumSet](https://docs.oracle.com/javase/8/docs/api/java/util/EnumSet.html) | [Set](https://docs.oracle.com/javase/8/docs/api/java/util/Set.html) | Высокопроизводительная реализация Set интерфейса, основанная на битовом векторе. Все элементы EnumSet объекта должны принадлежать к одному единственному enum типу | S/8 |

\* — размер дан в байтах для 32 битных систем и Compressed Oops, где С это capacity списка, S это size списка  
  
Узкоспециализированные коллекции на основе Set

|  |  |  |
| --- | --- | --- |
| **Название** | **Основан на** | **Описание** |
| [JobStateReasons](https://docs.oracle.com/javase/8/docs/api/javax/print/attribute/standard/JobStateReasons.html) | [HashSet](https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html) | Коллекция для хранения информации о заданиях печати (print job's attribute set). Узкоспециализированная коллекция основанная на HashSet с несколькими дополнительными методами |

Коллекции из пакета java.util.concurrent

|  |  |  |
| --- | --- | --- |
| **Название** | **Основан на** | **Описание** |
| [CopyOnWriteArraySet](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CopyOnWriteArraySet.html) | [Set](https://docs.oracle.com/javase/8/docs/api/java/util/Set.html) | Аналогично CopyOnWriteArrayList при каждом изменении создает копию всего множества, поэтому рекомендуется при очень редких изменениях коллекции и требованиях к thread-safe |
| [ConcurrentSkipListSet](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentSkipListSet.html) | [Set](https://docs.oracle.com/javase/8/docs/api/java/util/Set.html) | Является многопоточным аналогом TreeSet |

6) Коллекции, реализующие интерфейс **Map** (ассоциативный массив)

Коллекции, реализующие Map интерфейс

Универсальные коллекции общего назначения, реализующие Map:

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Основан на** | **Описание** | **Размер\*** |
| [HashMap](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html) | [Map](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html) | Реализация Map интерфейса с помощью хеш-таблиц (работает как не синхронная Hashtable, с поддержкой ключей и значений равных null). В большинстве случаев лучшая по производительности и памяти реализация Map интерфейса. Если интересны подробности устройства HashMap советую посмотреть эту [статью](http://habrahabr.ru/post/128017/). | 32 \* S + 4\*C |
| [LinkedHashMap](https://docs.oracle.com/javase/8/docs/api/java/util/LinkedHashMap.html) | [HashMap](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html) | Реализация Map интерфейса, на основе HashMap-таблицы с добавлением таблички-“связного списка”, для возможности обхода ключей в порядке добавления, а не как в HashMap по индексам. Данная коллекция работает почти так же быстро как HashMap. Так же она может быть полезна для создания кешей (смотрите removeEldestEntry(Map.Entry) ). Если интересны подробности устройства LinkedHashMap советую посмотреть эту [статью](http://habrahabr.ru/post/129037/). | 40 \* S + 4\*C |
| [TreeMap](https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html) | [NavigableMap](https://docs.oracle.com/javase/8/docs/api/java/util/NavigableMap.html) | Реализация NavigableMap с помощью красно-черного дерева, то есть при обходе коллекции, ключи будут отсортированы по порядку, так же NavigableMap позволяет искать ближайшее значение к ключу. | 40 \* S |
| [WeakHashMap](https://docs.oracle.com/javase/8/docs/api/java/util/WeakHashMap.html) | [Map](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html) | Аналогична HashMap, однако все ключи являются [слабыми ссылками](http://docs.oracle.com/javase/8/docs/api/java/lang/ref/WeakReference.html) (weak references), то есть garbage collected может удалить объекты ключи и объекты значения, если других ссылок на эти объекты не существует. WeakHashMap один из самых простых способов для использования всех преимуществ слабых ссылок. | 32 \* S + 4\*C |
| [EnumMap](https://docs.oracle.com/javase/8/docs/api/java/util/EnumMap.html) | [Map](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html) | Высокопроизводительная реализация Map интерфейса, основанная на простом массиве. Все ключи в этой коллекции могут принадлежать только одному enum типу. | 4\*C |
| [IdentityHashMap](https://docs.oracle.com/javase/8/docs/api/java/util/IdentityHashMap.html) | [Map](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html) | Identity-based Map, так же как HashMap, основан на хеш-таблице, однако в отличии от HashMap он никогда не сравнивает объекты на equals, только на то является ли они реально одиним и тем же объектом в памяти. Это во-первых, сильно ускоряет работу коллекции, во-вторых, полезно для защиты от «spoof attacks», когда сознательно генерируется объекты equals другому объекту. В-третьих, у данной коллекции много применений при обходе графов (таких как глубокое копирование или сериализация), когда нужно избежать обработки одного объекта несколько раз. | 8\*C |

\* — размер дан в байтах для 32 битных систем и Compressed Oops, где С это capacity списка, S это size списка

Коллекции из пакета java.util.concurrent

|  |  |  |
| --- | --- | --- |
| **Название** | **Основан на** | **Описание** |
| [ConcurrentHashMap](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html) | [ConcurrentMap](https://docs.oracle.com/javase/8/docs/api/java/util/ConcurrentMap.html) | Многопоточный аналог HashMap. Все данные делятся на отдельные сегменты и блокируются только отдельные сегменты при изменении, что позволяет значительно ускорить работу в многопоточном режиме. Итераторы никогда не кидают ConcurrentModificationException для данного вида коллекции |
| [ConcurrentSkipListMap](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentSkipListMap.html) | [ConcurrentNavigableMap](https://docs.oracle.com/javase/8/docs/api/java/util/ConcurrentNavigableMap.html) | Является многопоточным аналогом TreeMap |

7) Коллекции, основанные на интерфейсах **Queue/Deque** (очереди)

Коллекции, основанные на Queue/Deque

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Основан на** | **Описание** | **Размер\*** |
| [ArrayDeque](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayDeque.html) | [Deque](https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html) | Эффективная реализация Deque интерфейса, на основе динамического массива, аналогичная ArrayList | 6\*N |
| [LinkedList](https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html) | [Deque](https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html) | Реализация List и Deque интерфейса на основе двухстороннего связанного списка, то есть когда каждый элемент, указывает на предыдущий и следующий элемент.При работе через Queue интерфейс, LinkedList действует как FIFO очередь. | 40\*N |
| [PriorityQueue](https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html) | [Queue](https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html) | Неограниченная priority queue, основанная на куче (нeap). Элементы отсортированы в натуральном (естественном, для служебных типов перменных – типа String и т.д.) порядке или используя Comparator. Не может содержать null элементы. |  |

\* — размер дан в байтах для 32 битных систем и Compressed Oops, где С это capacity списка, S это size списка  
  
Многопоточные Queue и Deque, который определены в java.util.concurrent, требуют отдельной статьи, поэтому здесь приводить их не буду, если вам интересна информация о них советую прочитать вот эту [статью](http://habrahabr.ru/company/luxoft/blog/157273/)

8) Прочие коллекции

Прочие коллекции

|  |  |  |
| --- | --- | --- |
| **Название** | **Описание** | **Размер\*** |
| [BitSet](http://docs.oracle.com/javase/8/docs/api/java/util/BitSet.html) | Несмотря на название, BitSet не реализует интерфейс Set. BitSet служит для компактной записи массива битов. | N / 8 |

9) Методы работы с коллекциями

Методы работы с коллекциями

Алгоритмы в классе Collections содержится много полезных статистических методов.  
**Для работы с любой коллекцией:**

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| **frequency**(Collection, Object) | Возвращает количество вхождений данного элемента в указанной коллекции |
| **disjoint**(Collection, Collection) | Возвращает true, если в двух коллекциях нет общих элементов |
| **addAll**(Collection<? super T>, T...) | Добавляет все элементы из указанного массива (или перечисленные в параметрах) в указанную коллекцию |
| **min**(Collection) | Возвращение минимального элемента из коллекции |
| **max**(Collection) | Возвращение максимального элемента из коллекции |

**Для работы со списками:**

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| **sort**(List) | Сортировка с использованием алгоритма сортировки соединением (merge sort algorithm), производительность которой в большинстве случаев близка к производительности быстрой сортировки (high quality quicksort), гарантируется O(n\*log n) производительность (в отличии от quicksort), и стабильность (в отличии от quicksort). Стабильная сортировка это такая которая не меняет порядок одинаковых элементов при сортировке |
| **binarySearch**(List, Object) | Поиск элемента в списке (list), используя binary search алгоритм. |
| **reverse**(List) | Изменение порядка всех элементов списка (list) |
| **shuffle**(List) | Перемешивание всех элементов в списке в случайном порядке |
| **fill**(List, Object) | Переписывание каждого элемента в списке каким-либо значением |
| **copy**(List dest, List src) | Копирование одного списка в другой |
| **rotate**(List list, int distance) | Передвигает все элементы в списке на указанное расстояние |
| **replaceAll**(List list, Object oldVal, Object newVal) | Заменяет все вхождения одного значения на другое |
| **indexOfSubList**(List source, List target) | Возвращает индекс первого вхождения списка target в список source |
| **lastIndexOfSubList**(List source, List target) | Возвращает индекс последнего вхождения списка target в список source |
| **swap**(List, int, int) | Меняет местами элементы, находящиеся на указанных позициях |

В Java 8 так же появился такой способ работы с коллекциями как stream Api, но мы рассмотрим примеры его использование далее в разделе 5.

10) Как устроенны разные типы коллекций JDK внутри

Как устроенны разные типы коллекций JDK внутри

|  |  |
| --- | --- |
| **Коллекция** | **Описание внутреннего устройства** |
| ArrayList | Данная коллекция лишь надстройка над массивом + переменная хранящая size списка. Внутри просто массив, который пересоздается каждый раз когда нет места для добавления нового элемента. В случае, добавления или удаления элемента внутри коллекции весь хвост сдвигается в памяти на новое место. К счастью, копирование массива при увеличении емкости или при добавлении/удалении элементов производится быстрыми нативными/системными методами. Если интересны подробности  советую посмотреть эту [статью](http://habrahabr.ru/post/128269/). |
| LinkedList | Внутри коллекции используется внутренний класс Node, содержащий ссылку на предыдущий элемент, следующий элемент и само значение элемента. В самом инстансе коллекции хранится размер и ссылки на первый и последний элемент коллекции. Учитывая что создание объекта дорогое удовольствие для производительности и затратно по памяти, LinkedList чаще всего работает медленно и занимает намного больше памяти чем аналоги. Обычно ArrayList, ArrayDequery лучшее решение по производительности и памяти, но в некоторых редких случаях (частые вставки в середину списка с редкими перемещениями по списку), он может быть быть полезен (но в этом случае полезнее использовать TreeList от apache). Если интересны подробности советую посмотреть эту [статью](http://habrahabr.ru/post/127864/). |
| HashMap | Данная коллекция построена на хеш-таблице, то есть внутри коллекции находится массив внутреннего класса (buket) Node равный capacity коллекции. При добавлении нового элемента вычисляться его хеш-функция, делиться на capacity HashMap по модулю и таким образом вычисляется место элемента в массиве. Если на данном месте ещё не храниться элементов создается новый объект Node с ссылкой на добавляемый элемент и записывается в нужное место массива. Если на данном месте уже есть элемент/ы (происходит хеш-коллизия), то так Node является по сути односвязным списком, то есть содержит ссылку на следующий элемент, то можно обойти все элементы в списке и проверить их на equals добавляемому элементу, если этот такого совпадение не найдено, то создается новый объект Node и добавляется в конец списка. В случае, если количество элементов в связном списке (buket) становится более 8 элементов, вместо него создается бинарное дерево. Подробнее о хеш таблицах смотрите [вики](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D0%B0) (в HashMap используется метод цепочек для разрешения коллизий). Если интересны подробности устройства HashMap советую посмотреть эту [статью](http://habrahabr.ru/post/128017/). |
| HashSet | HashSet это просто HashMap, в которую записывается фейковый объект Object вместо значения, при этом имеет значение только ключи. Внутри HashSet всегда хранится коллекция HashMap. |
| IdentityHashMap | IdentityHashMap это аналог HashMap, но при этом не требуется элементы проверять на equals, так как разными считаются любые два элементы. указывающие на разные объекты. Благодаря этому удалось избавиться от внутреннего класса Node, храня все данные в одном массиве, при этом при коллизиях ищется подходящая свободная ячейка до тех пор пока не будет найдена ([метод открытой адресации](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D0%B0#.D0.9E.D1.82.D0.BA.D1.80.D1.8B.D1.82.D0.B0.D1.8F_.D0.B0.D0.B4.D1.80.D0.B5.D1.81.D0.B0.D1.86.D0.B8.D1.8F)).Подробнее о хеш таблицах смотрите [вики](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D0%B0) (в IdentityHashMap используется метод открытой адресации для разрешения коллизий) |
| LinkedHashMap/LinkedHashSet | Внутренняя структура практически такая же как при HashMap, за исключением того что вместо внутреннего класса Node, используется TreeNode, которая знает предыдущее и следующее значение, это позволяет обходить LinkedHashMap по порядку добавления ключей. По сути, LinkedHashMap = HashMap + LinkedList. Если интересны подробности устройства LinkedHashMap советую посмотреть эту [статью](http://habrahabr.ru/post/129037/).  Т.е. это как HashMap, только для каждого элемента добавились ссылки на предыдущий и последующий (по времени вставки) элементе (не цепочки, а всей таблицы). А при проходе по таблице итератором можно менять порядок просмотра (*свойство ‘assetOrder’*) |
| TreeMap/TreeSet | Внутренняя структура данных коллекций построена на сбалансированным красно-черным деревом, подробнее о нем можно почитать в [вики](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B0%D1%81%D0%BD%D0%BE-%D1%87%D1%91%D1%80%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE) |
| WeakHashMap | Внутри все организовано практически как в HashMap, за исключением что вместо обычных (*сильных*) ссылок (*которые являются значениями в HashMap таблицах, тогда как ключами в них являются хеши ключей*) используются WeakReference ссылки (*слабые ссылки, объекты которых удалаются в любой момент при вызове cборщика мусора и для этого не обязательно условие нехватки памяти*) и есть отдельная очередь ReferenceQueue (*ее функция в том, что когда объект доступный с помощью ссылки удаляется, то сам объект ссылки становится доступным в очереди ReferenceQueue. Этот механизм позволяет удалять пустые ссылки*). |
| CopyOnWriteArrayList/Set | При добавлении нового элемента создается новая таблица |
| ConcurrentHashMap | HashMap где все данные делятся на отдельные сегменты и блокируются только отдельные сегменты при изменении |
| [ConcurrentSkipListMap](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentSkipListMap.html)/Set | Используется механизм SkipList – вначале проваливание на нижний уровень, потом “орел/решка” для поднятия еще и на узловой уровень наверх |
| EnumSet/EnumMap | В EnumSet и EnumMap в отличие от HashSet и HashMap используются битовые векторы и массивы для компактного хранения данных и ускорения производительности. Ограничением этих коллекций является то что EnumSet и EnumMap могут хранить в качестве ключей только значения одного Enum'а. |

11) Другие полезные методы стандартной библиотеки коллекций:

1. **Wrapper implementations** – Обертки для добавления функциональности и изменения поведения других реализаций. Доступ исключительно через статистические методы.

* ***Collections.unmodifiableInterface***– Обертка для создания не модифицируемой коллекции на основе указанной, при любой попытки изменения данной коллекции будет выкинут UnsupportedOperationException
* ***Collections.synchronizedInterface***– Создания синхронизированной коллекции на основе указанной, до тех пор пока доступ к базовой коллекции идет через коллекцию-обертку, возвращенную данной функцией, потокобезопасность гарантируется.
* ***Collections.checkedInterface*** – Возвращает коллекцию с проверкой правильности типа динамически (во время выполнения), то есть возвращает type-safe view для данной коллекции, который выбрасывает ClassCastException при попытке добавить элемент ошибочного типа. При использовании механизма generic'ов JDK проверяет на уровне компиляции соответствие типов, однако этот механизм можно обойти, динамическая проверка типов не позволяет воспользоваться этой возможностью.

2. **Adapter implementations** – данная реализация адаптирует один интерфейс коллекций к другому

* ***newSetFromMap(Map)*** – Создает из любой реализации Set интерфейса реализацию Map интерфейса.
* ***asLifoQueue(Deque)*** - возвращает view из Deque в виде очереди работающей по принципу Last In First Out (LIFO).

3. **Convenience implementations** – Высокопроизводительные «мини-реализации» для интерфейсов коллекций.

* ***Arrays.asList*** – Позволяет отобразить массив как список (list)
* ***emptySet, emptyList и emptyMap*** – возвращает пустую не модифицированную реализацию empty set, list, or map
* ***singleton, singletonList и singletonMap*** – возвращает не модифицируемый set, list или map, содержащий один заданный объект (или одно связь ключ-значение)
* ***nCopies*** – Возвращает не модифицируемый список, содержащий n копий указанного объекта

4. **Абстрактные реализации интерфейсов** — Реализация общих функций (скелета коллекций) для упрощения создания конкретных реализаций коллекций.

* ***AbstractCollection*** – Абстрактная реализация интерфейса Collection для коллекций, которые не являются ни множеством, ни списком (таких как «bag» или multiset).
* ***AbstractSet*** — Абстрактная реализация Set интерфейса.
* ***AbstractList*** – Абстрактная реализация List интерфейса для списков, позволяющих позиционный доступ (random access), таких как массив.
* ***AbstractSequentialList*** – Абстрактная реализация List интерфейса, для списков, основанных на последовательном доступе (sequential access), таких как linked list.
* ***AbstractQueue*** — Абстрактная Queue реализация.
* ***AbstractMap*** — Абстрактная Map реализация.

5. **Инфраструктура**

* ***Iterators*** – Похожий на обычный Enumeration интерфейс, но с большими возможностями.
* ***Iterator*** – Дополнительно к функциональности Enumeration интерфейса, который включает возможности по удалению элементов из коллекции.
* ***ListIterator*** – это Iterator который используется для lists, который добавляет к функциональности обычного Iterator интерфейса, такие как итерация в обе стороны, замена элементов, вставка элементов, получение по индексу.

6. Ordering

* ***Comparable*** — используется только для сравнения объектов класса, в котором данный интерфейс реализован (содержит метод compareTo). Пишется в классе, объекты которого будут создаваться и перебираться в другом классе. Этот порядок может быть использован в методах сортировки или для реализации sorted set или map.
* ***Comparator*** — представляет отдельную реализацию и ее можно использовать многократно и с различными классами(содержит метод compare).

7. Runtime exceptions

* ***UnsupportedOperationException*** – Это ошибка выкидывается когда коллекция не поддерживает операцию, которая была вызвана.
* ***ConcurrentModificationException*** – Выкидывается iterators или list iterators, если коллекция на которой он основан была неожиданно (для алгоритма) изменена во время итерирования, также выкидывается во views основанных на списках, если главный список был неожиданно изменен.

8. Производительность

* ***RandomAccess*** — Интерфейс-маркер, который отмечает списки позволяющие быстрый (как правило за константное время) доступ к элементу по позиции. Это позволяет генерировать алгоритмы учитывающие это поведение для выбора последовательного и позиционного доступа.

9. Утилиты для работы с массивами

* ***Arrays*** – Набор статических методов для сортировки, поиска, сравнения, изменения размера, получения хеша для массива. Так же содержит методы для преобразования массива в строках и заполнения массива примитивами или объектами.