

Luxoft
Компаниярейтинг
106,62

Профиль

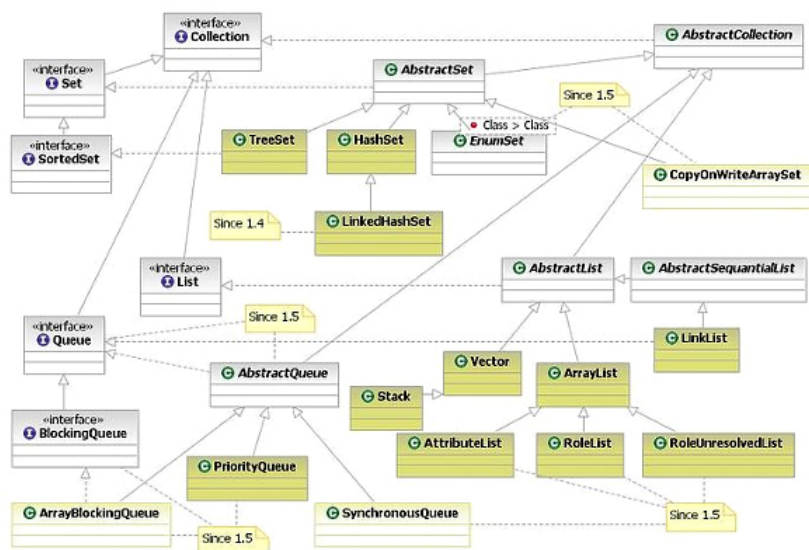
144
Блог0
Вакансии1,5k
Подписчики

27 октября 2015 в 23:03

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

Разработка веб-сайтов*, Программирование*, Алгоритмы*, Java*, Блог компании Luxoft

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'

1. JPA и Hibernate в вопросах и ответах
2. Триста пятьдесят самых популярных не мобильных Java opensource проектов на github
3. Коллекции в Java (стандартные, guava, apache, trove, gs-collections и другие)
4. Java Stream API
5. Двести пятьдесят русскоязычных обучающих видео докладов и лекций о Java
6. Список полезных ссылок для Java программиста
7. Типовые задачи
 - 7.1 Оптимальный путь преобразования InputStream в строку
 - 7.2 Самый производительный способ обхода Map'ы, подсчет количества вхождений подстроки
8. Библиотеки для работы с Json (Gson, Fastjson, LoganSquare, Jackson, JsonPath и другие)

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	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	Основной интерфейс для большинства коллекций (кроме Map)
List	Список это упорядоченная в порядке добавления коллекция, так же известная как последовательность (sequence). Дублирующие элементы в большинстве реализаций — разрешены. Позволяет доступ по индексу элемента. Расширяет Collection интерфейс.
Set	Интерфейс реализующий работу с множествами (похожими на математические множества), дублирующие элементы запрещены. Может быть, а может и не быть упорядоченным. Расширяет Collection интерфейс.
Queue	Очередь — это коллекция, предназначенная для хранения объектов до обработки, в отличие от обычных операций над коллекциями, очередь предоставляет дополнительные методы добавления, получения и просмотра. Быстрый доступ по индексу элемента, как правило, не содержит. Расширяет Collection интерфейс
Deque	Двухсторонняя очередь, поддерживает добавление и удаление элементов с обоих концов. Расширяет Queue интерфейс.
Map	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	Автоматически отсортированное множество, либо в натуральном порядке (для подробностей смотрите Comparable интерфейс), либо используя Comparator. Расширяет Set интерфейс
SortedMap	Это map'a ключи которой автоматически отсортированы, либо в натуральном порядке, либо с помощью comparator'a. Расширяет Map интерфейс.
NavigableSet	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	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	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	Эта многопоточная очередь может блокировать вставляющий поток, до тех пор, пока принимающий поток не вытаскивает элемент из очереди, таким образом с её помощью можно реализовывать синхронные и асинхронные передачи сообщений между потоками
BlockingDeque	Аналогично BlockingQueue, но для двухсторонней очереди
ConcurrentMap	Интерфейс, расширяет интерфейс Map. Добавляет ряд новых атомарных методов: putIfAbsent, remove, replace, которые позволяют облегчить и сделать более безопасным многопоточное программирование.
ConcurrentNavigableMap	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](#).

2) Таблица с очень кратким описанием всех коллекций

▼ [Таблица с очень кратким описанием всех коллекций](#)

Тип	Однопоточные	Многopоточные
Lists	<ul style="list-style-type: none"> ArrayList — основной список, основан на массиве LinkedList — полезен лишь в некоторых редких случаях Vector — устарел 	<ul style="list-style-type: none"> CopyOnWriteArrayList — редкие обновления, частые чтения
Queues / Deques	<ul style="list-style-type: none"> ArrayDeque — основная реализация, основан на массиве Stack — устарел PriorityQueue — отсортированная очередь 	<ul style="list-style-type: none"> ArrayBlockingQueue — блокирующая очередь на связанных нодах ConcurrentLinkedDeque / ConcurrentLinkedQueue — очередь на связанных нодах 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	<ul style="list-style-type: none"> 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 	<ul style="list-style-type: none"> ConcurrentHashMap — основная многопоточная реализация ConcurrentSkipListMap — отсортированная многопоточная реализация
Sets	<ul style="list-style-type: none"> HashSet — основная реализация множества EnumSet — множество из enums BitSet* — множество битов LinkedHashSet — сохраняет порядок вставки TreeSet — отсортированные set 	<ul style="list-style-type: none"> 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	Изначально задумывался как синхронизированный аналог HashMap, когда ещё не было возможности получить версию коллекции используя Collections.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	Раньше использовался как синхронный вариант ArrayList, однако устарел по тем же причинам что и Hashtable.
Stack	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	Hashtable	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 .
UIDefaults	Hashtable	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	Основа	Описание	Размер*
ArrayList	List	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 .	4*N
LinkedList	List	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 .	24*N

Коллекции из пакета java.util.concurrent

Название	Основа	Описание
CopyOnWriteArrayList	List	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	Основана	Описание
RoleList	ArrayList	Коллекция для хранения списка ролей (Roles). Узкоспециализированная коллекция основанная на ArrayList с несколькими дополнительными методами
RoleUnresolvedList	ArrayList	Коллекция для хранения списка unresolved ролей (Unresolved Roles). Узкоспециализированная коллекция основанная на ArrayList с несколькими дополнительными методами
AttributeList	ArrayList	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	Основана	Описание	Размер*
HashSet	Set	Реализация Set интерфейса с использованием хеш-таблиц. В большинстве случаев, лучшая возможная реализация Set интерфейса.	32*S + 4*C
LinkedHashSet	HashSet	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	NavigableSet	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 основано на HashMap	40 * S
EnumSet	Set	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 C это capacity списка, S это size списка

Узкоспециализированные коллекции на основе Set

Название	Основана	Описание
JobStateReactions	HashSet	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	Основана	Описание
CopyOnWriteArraySet	Set	Аналогично CopyOnWriteArrayList при каждом изменении создает копию всего множества, поэтому рекомендуется при очень редких изменениях коллекции и требованиях к thread-safe
ConcurrentSkipListSet	Set	Является многопоточным аналогом TreeSet

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

▼ Коллекции, реализующие [Map](#) интерфейс

Универсальные коллекции общего назначения, реализующие Map:

Название	Основана	Описание	Размер*
HashMap	Map	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 .	32 * S + 4*C
LinkedHashMap	HashMap	Реализация Map интерфейса, на основе хеш-таблицы и связанного списка, то есть ключи в Map'e хранятся и обходятся во порядке добавления. 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 .	40 * S + 4*C
TreeMap	NavigableMap	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	Map	Аналогична HashMap, однако все ключи являются слабыми ссылками (weak references), то есть garbage collected может удалить объекты ключи и 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	Map	Высокопроизводительная реализация Map интерфейса, основанная на простом массиве. Все ключи в этой коллекции могут принадлежать только одному enum типу.	4*C
IdentityHashMap	Map	Identity-based Map, так же как HashMap, основан на хеш-таблице, однако в отличие от HashMap он никогда не сравнивает объекты на equals, только на то является ли они реально одним и тем же объектом в памяти. Это во-первых, сильно ускоряет работу коллекции, во-вторых, полезно для защиты от «spoof attacks», когда сознательно генерируются 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 C это capacity списка, S это size списка

Коллекции из пакета java.util.concurrent

Название	Основа на	Описание
ConcurrentHashMap	ConcurrentMap	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	ConcurrentNavigableMap	Является многопоточным аналогом TreeMap

7) Коллекции, основанные на интерфейсах **Queue/Deque** (очереди)▼ [Коллекции, основанные на Queue/Deque](#)

Название	Основана на	Описание	Размер*
ArrayDeque	Deque	Эффективная реализация Deque интерфейса, на основе динамического массива, аналогичная ArrayList	6*N
LinkedList	Deque	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	Queue	Неограниченная priority queue, основанная на куче (heap). Элементы отсортированы в натуральном порядке или используя Comparator. Не может содержать null элементы.	

* — размер дан в байтах для 32 битных систем и CompressedOops, где C 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](#)

8) Прочие коллекции

▼ [Прочие коллекции](#)

Название	Описание	Размер*
BitSet	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 отличие от quicksort), и стабильность (в отличие 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 использования далее в разделе 5.

10) Как устроены разные типы коллекций JDK внутри

▼ [Как устроены разные типы коллекций JDK внутри](#)

Коллекция	Описание внутреннего устройства
ArrayList	Данная коллекция лишь настройка 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 советуем посмотреть эту статью .
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, ArrayDeque лучше решение по производительности и памяти, но в некоторых редких случаях (частые вставки в середину списка с редкими перемещениями по списку), он может быть полезен (но в этом случае полезнее использовать TreeList от apache). Если интересны подробности советуем посмотреть эту статью .
HashMap	Данная коллекция построена на хеш-таблице, то есть внутри коллекции находится массив внутреннего класса (bucket) Node равный capacity коллекции. При добавлении нового элемента вычисляется его хеш-функция, делится на capacity HashMap по модулю и таким образом вычисляется место элемента в массиве. Если на данном месте еще не хранятся элементы 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 этого совпадения не найдено, то создается новый объект Node и добавляется в конец списка. В случае, если количество элементов в связном списке (bucket) становится более 8 элементов, вместо него создается бинарное дерево. Подробнее о хеш-таблицах смотрите вики (в HashMap используется метод цепочек для разрешения коллизий). Если интересны подробности устройства HashMap советуем посмотреть эту статью .
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 элемента, 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). Подробнее о хеш таблицах смотрите вики (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 .
TreeMap/TreeSet	Внутренняя структура данных коллекций построена на сбалансированном красно-черным деревом, подробнее о нем можно почитать в вики
WeakHashMap	Внутри все организовано практически как в HashMap, за исключением что вместо обычных ссылок используются WeakReference и есть отдельная очередь ReferenceQueue, необходимая для удаления WeakEntries
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'a.

11) Другие полезные сущности стандартной библиотеки коллекций

▼ Другие полезные сущности стандартной библиотеки коллекций

Давайте посмотрим какие ещё полезные сущности содержит [официальный гайд по коллекциям](#)

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 попытке 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.

4) **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.

5) Ordering

- **Comparable** — Определяет натуральный порядок сортировки для классов, которые реализуют их. Этот порядок может быть использован в методах сортировки или для реализации sorted set или map.
- **Comparator** — Represents an order relation, which can be used to sort a list or maintain order in a sorted set or map. Can override a type's natural ordering or order objects of a type that does not implement the Comparable interface.

6) 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.

7) 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.

8) 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 строку и заполнения массива примитивами или объектами.

II. Краткий обзор сторонних библиотек коллекций

Итак, Я хотел бы сделать обзор следующих сторонних библиотек: guava, apache, trove и gs-collections. Почему именно эти библиотеки? Guava и Apache Commons Collections очень популярны и встречались мне почти в любом Java проекте, Trove — тоже очень популярная библиотека, когда нужно уменьшить память и улучшить производительность работы с коллекциями. GS-collections — судя по оценкам весьма популярная библиотека на github'e (>1300 звезд), больше неё набрала «звезд» только guava. Так же мельком захвачу несколько других популярных библиотек.

Итак, для начала рассмотрим что предлагают различные библиотеки, их главные фишки (Предупреждение: это очень субъективно, для кого-то главными фишками будут совсем другие возможности библиотек).

2.1 Фишки разных библиотек коллекций

Давайте сделаем небольшой обзор главных фишек (на мой взгляд) разных библиотек коллекций:

1) **Guava** — данная коллекция от гугл практически самая популярная после стандартного фреймворка коллекций, она добавляет ряд интересных коллекций, но самая главная «фишка» это скорее богатый набор классов-утилит со статическими методами, расширяющими возможности Collections для работы со стандартными коллекциями, чем новые виды коллекций. Стандартные коллекции она практически не заменяет.

2) **Apache Commons Collections** — данная коллекция ближайший «конкурент» guava, она так же предоставляет ряд интересных коллекций, утилит по работе со стандартными коллекциями Java, а так же большое количество wrapper'ов для изменения поведения коллекций. Кроме того она предоставляет свою реализацию map'ы с более простым механизмом итерирования по ней.

3) **Trove** — фишка данной коллекции в первую очередь в производительности и сокращении памяти, поэтому она предлагает более быстрые реализации стандартных коллекций (и требующие меньше памяти), а так же коллекции примитивных типов.

4) **GS-collections** — фишка данной коллекции в идее объединить методы обработки, такие как сортировка и классы коллекций для создания замены использования статических методов классов-утилит. Данная библиотека предлагает замену практически всех стандартным коллекциям и добавляет несколько новых.

III. Альтернативные виды коллекций в разных библиотеках

Тут я попробую кратко рассмотреть, какие новые интересные виды коллекций можно найти в разных библиотеках:

3.1 Альтернативные виды коллекций у Guava

Официальная информация: [документация](#), [исходные коды](#), [javadoc](#).

Как подключить к проекту:

▼ [Maven, Gradle](#)

Maven

```
<dependency>
  <groupId>com.google.guava</groupId>
  <artifactId>guava</artifactId>
  <version>18.0</version>
</dependency>
```

Gradle

```
dependencies {
  compile 'com.google.guava:guava:18.0'
}
```

Гугл разработал ряд интересных дополнений к существующим коллекциям, которые весьма и весьма полезны, если вы можете использовать guava библиотеку в своем проекте. Можно сказать, что эти коллекции давно уже стали стандартом де-факто для большинства Java проектов, поэтому для любого опытного разработчика важно знать их, даже если он по каким-то причинам не может использовать их в своих проектах, зачастую на собеседованиях можно услышать вопросы по guava коллекциям.

Давайте рассмотрим их поподробнее. Для начала рассмотрим интерфейсы основных коллекций и группы классов в guava.

Внимание: если таблица не помещается целиком, попробуйте уменьшить масштаб страницы или открыть в другом браузере.

Название	Описание	Примеры реализаций	Примеры использования
ImmutableCollection ImmutableList ImmutableSet ... и т.д.	Хотя в стандартном фреймворке Java коллекций есть возможность сделать коллекцию неизменяемой вызвав <code>Collections.unmodifiableCollection</code> (<code>unmodifiableList</code> или <code>unmodifiableMap</code>), но этот подход не самый оптимальный, так как отдельный тип для неизменяемых коллекций позволяет быть уверенным, что это коллекция действительно неизменяемая, вместо ошибок времени исполнения, при попытке изменить коллекцию, будут ошибки во время компиляции проекта, к тому же в стандартном фреймворке коллекций Java неизменяемые коллекции по-прежнему тратят ресурсы на поддержку синхронизации при многопоточном чтении и т. п. операциях, в то время как <code>ImmutableCollection</code> guava «знают» что они неизменяемые и оптимизированы с учетом этого.	JDK: ImmutableCollection , ImmutableList , ImmutableSet , ImmutableSortedSet , ImmutableMap , ImmutableSortedMap Guava: ImmutableMultiset , ImmutableSortedMultiset , ImmutableMultimap , ImmutableListMultimap , ImmutableSetMultimap , ImmutableBiMap , ImmutableClassToInstanceMap , ImmutableTable	— если публичный метод возвращает коллекцию, которую гарантировано не должны менять другие классы, — если известно что значения коллекции больше никогда не должны меняться
Multiset	Коллекция аналогичная <code>Set</code> , но позволяющая дополнительно считать количество добавлений элемента. Очень полезна для тех задач, когда нужно не только знать есть ли данный элемент в данном множестве, но и посчитать их количество (самый простой пример подсчет количества упоминаний тех или иных слов в каком-либо тексте). То есть данная коллекция более удобный вариант коллекции <code>Map<T, Integer></code> , с методами специально предназначенными для подобных коллекций, позволяет очень сильно сократить количество лишнего кода в таких случаях.	HashMultiset , TreeMultiset , LinkedHashMultiset , ConcurrentHashMultiset , ImmutableMultiset SortedMultiset	— подсчет кол-ва вхождений слов в тексте — подсчет кол-ва букв в тексте — подсчет кол-ва любых объектов
Multimap	Практически любой опытный Java разработчик сталкивался с необходимостью использовать структуры вроде <code>Map<K, List<V>></code> или <code>Map<K, Set<V>></code> , при этом приходилось писать много лишнего кода, для упрощения работы в библиотеку guava были введены <code>Multimap</code> , то есть коллекции, позволяющие просто работать со случаями когда один ключ и много значений у этого ключа. В отличие от конструкций вроде <code>Map<K, Set<V>></code> , <code>Multimap</code> предоставляет ряд удобных функций для сокращения кода и упрощения алгоритмов.	ArrayListMultimap , HashMultimap , TreeMultimap , LinkedHashMultimap , TreeMultimap , ImmutableListMultimap , ImmutableSetMultimap	— реализация отношений один ко многим, таких как: учитель — ученики отдел — работники начальник — подчиненные
BiMap	Достаточно часто встречаются ситуации, когда требуется создать <code>Map</code> 'у работающую в обе стороны, то есть когда ключ и значение могут меняться местами (например, русско-английский словарь, когда в одном случае	HashBiMap , ImmutableBiMap , EnumBiMap , EnumHashBiMap	— словарь для перевода с одного языка в другой и обратно,

	требуется получить по русскому слову — английское, в другом наоборот по английскому-русское). Обычно, это решается созданием двух Map, где в одной будет ключ1-ключ2, в другой ключ2-ключ1). BiMap позволяет решить эту задачу с помощью лишь одной коллекции. К тому же это исключает проблемы и ошибки синхронизации при использовании двух коллекций.		— любая конвертация данных в обе стороны,
Table	Эта коллекция служит для замены коллекций вида Map<FirstName, Map<LastName, Person>>, которые неудобны в использовании.	HashBasedTable, TreeBasedTable, ImmutableTable, ArrayTable	— таблица, например, как в Excel — любые сложные структуры данных с большим количеством столбцов,
ClassToInstanceMap	Иногда нужно хранить в Map'e не ключ-значение, а тип-значение этого типа, для этого служит данная коллекция. То есть это технически это более удобный и безопасный аналог Map<Class<? extends B>, B>	MutableClassToInstanceMap, ImmutableClassToInstanceMap.	
RangeSet	Коллекция для хранения разных открытых и закрытых отрезков числовых значений, при этом отрезки могут объединяться с друг другом.	ImmutableRangeSet, TreeRangeSet	Геометрические отрезки Временные отрезки
RangeMap	Коллекция, похожая на RangeSet, но при этом отрезки никогда не объединяются друг с другом.	ImmutableRangeMap, TreeRangeMap	Геометрические отрезки Временные отрезки
LoadingCache	Коллекция, похожая на ConcurrentMap, но при этом можно указать время какое будет храниться каждый элемент. Очень удобная коллекция для организации кэшей, подсчета количества ввода ошибочных паролей за какой-то промежуток времени и т.п. задач	ForwardingLoadingCache, ForwardingLoadingCache, SimpleForwardingLoadingCache	кэши, хранение ошибочных попыток ввода пароля и т.п.

3.2 Новые виды коллекций из Apache Commons Collections

Официальная информация: [документация](#), [исходные коды](#), [документация пользователя](#), [javadoc](#).

Как подключить к проекту:

▼ [Maven](#), [Gradle](#), [Ivy](#)

Maven

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-collections4</artifactId>
  <version>4.0</version>
</dependency>
```

Gradle

```
'org.apache.commons:commons-collections4:4.0'
```

Ivy

```
<dependency org="org.apache.commons" name="commons-collections4" rev="4.0"/>
```

Внимание: если таблица не помещается целиком, попробуйте уменьшить масштаб страницы или открыть в другом браузере.

Название	Описание	Примеры реализаций	Примеры использования
Unmodifiable	Интерфейс аналогичный Immutable классам guava	UnmodifiableBag, UnmodifiableBidiMap, UnmodifiableCollection, UnmodifiableList, UnmodifiableMap и т.п.	во всех случаях когда нужно создать не модифицированную коллекцию
IterableMap	Аналог интерфейса Map, но позволяющий итерироваться по Map напрямую без создания entry set. Используется почти во всех реализациях Map в данной библиотеке.	HashMap, LinkedMap, ListOrderedMap и ряд других	Такие же как у обычной map'ы
OrderedMap	Позволяет создавать Map'ы, упорядоченные по порядку	LinkedMap,	В случаях, когда обычно

	добавления, но не использующие сортировку	ListOrderedMap	используется отдельно List и отдельно Map'a
BidiMap	Аналог BiMap из Guava, то есть возможность получать значение по ключу, так и ключ по значению	TreeBidiMap , DualHashBidiMap , DualLinkedHashBidiMap , DualTreeBidiMap и т.п.	Любые конвертации один к одному, которые требуется выполнять в обе стороны
Bags	Аналог Multiset из Guava, то есть возможность сохранять количество элементов каждого типа	CollectionBag , HashBag , SynchronizedBag , TreeBag и другие	подсчет кол-ва любых объектов
BoundedCollection, BoundedMap	Позволяет создавать динамические коллекции, ограниченные каким-то размером сверху	CircularFifoQueue , FixedSizeList , FixedSizeMap , LRUMap	в случае, когда вы точно знаете что в коллекции не может быть больше определенного количества элементов
MultiMap	Аналог Multimap из Guava, то есть возможность сохранять множество элементов для одного ключа	MultiValueMap	для коллекций со связями один ключ – много значений
Trie	Коллекция для создания и хранения упорядоченных деревьев	PatriciaTrie	создание деревьев
TreeList	Замена ArrayList и LinkedList, если требуется вставить элемент в середину списка, так как в данном списке данные хранятся в виде дерева, что позволяет с одной стороны относительно быстро получать данные по индексу, с другой стороны быстро вставлять данные в середину списка.	TreeList	замена LinkedList при частых добавлениях/удалениях в середине списка

3.3 Trove коллекции

В отличие от остальных библиотек альтернативных коллекций, Trove не предлагает никакие новые уникальные виды коллекций, зато предлагает оптимизацию существующих:

Во-первых, как известно, примитивные типы Java нельзя добавить в стандартные коллекции, только их обертки, что резко увеличивает занимаемую память и несколько ухудшает производительность коллекций. Trove предлагает набор коллекций, ключи и значения которых могут содержать примитивные типы.

Во-вторых, стандартные коллекции часто реализованы не самым оптимальным способом по потреблению памяти, например, каждый элемент HashMap храниться в отдельном объекте, а HashSet это HashMap хранящая фейковые объекты вместо ключей. Trove предлагает свои реализации таких коллекций на основе массивов и открытой адресации, что позволяет значительно сократить требуемую память и в некоторых случаях улучшить производительность.

Update: В комментариях, к статье было высказано мнение что Trove плохо использовать в новых проектах, так как он по всех параметрам уступает fastutil или GS (кол-во багов, полнота покрытия интерфейсов, производительность, активность поддержки, и т. д.). К сожалению, у меня нет возможности сейчас провести полноценный анализ/сравнение Trove с fastutil и GS, поэтому не могу проверить данное мнение, просто учитывайте его при выборе библиотеки альтернативных коллекций.

Официальная информация: [документация](#), [исходные коды](#), [javadoc](#).

Как подключить к проекту:

▼ [Maven](#), [Gradle](#), [Ivy](#)

Maven

```
<dependency>
  <groupId>net.sf.trove4j</groupId>
  <artifactId>trove4j</artifactId>
  <version>3.0.3</version>
</dependency>
```

Gradle

```
'net.sf.trove4j:trove4j:3.0.3'
```

Ivy

```
<dependency org="net.sf.trove4j" name="trove4j" rev="3.0.3"/>
```

Название	Аналог JDK	Описание
THashMap	HashMap	Реализация Map интерфейса, которая использует хеш-таблицу с алгоритмом "открытой адресации" для разрешения коллизий (в отличие от HashMap где используется метод цепочек). Это позволяет не хранить и не создавать объекты класса Node, при этом сильно экономится память и, в некоторых случаях, улучшается производительность.
THashSet	HashSet	Реализация Set интерфейса, которая использует хеш-таблицу с алгоритмом "открытой адресации" для разрешения коллизий
TLinkedHashSet	LinkedHashSet	Аналог LinkedHashSet, но используя хеш-таблицы с алгоритмом "открытой адресации"
TLinkedList	LinkedList	Более производительный аналог связанного списка, однако накладывающий ряд ограничений на данные.
TByteArrayList, TIntArrayList и т.п.	ArrayList	Аналог ArrayList, который непосредственно хранит примитивные числовые значения, что резко сокращает затраты памяти и ускоряет обработку. Есть коллекции для всех семи примитивных числовых типов, шаблон наименования T[Тип]ArrayList
TCharLinkedList, TFloatLinkedList и т.п.	LinkedList	Аналог LinkedList для хранения семи примитивных числовых типов, шаблон наименования T[Тип]LinkedList
TByteArrayStack, TLongArrayStack	ArrayDeque	Реализация стека для хранения примитивных числовых типов, шаблон наименования T[Тип]LinkedList
TIntQueue, TCharQueue	ArrayDeque	Реализация очереди для хранения примитивных числовых типов, шаблон наименования T[Тип]Queue
TShortHashSet, TDoubleHashSet	HashSet	Реализация Set интерфейса для хранения примитивных типов, с алгоритмом открытой адресации, шаблон наименования T[Тип]HashSet
TLongLongHashMap, TFloatObjectHashMap, TShortObjectHashMap и т.п.	HashMap	Реализация Map интерфейса для хранения примитивных типов, с алгоритмом открытой адресации, шаблон наименования T[Тип][Тип]HashMap, где тип может быть Object

3.4 GS-collections коллекции

Основная фишка данной библиотеке в том что нелогично и некрасиво то что методы обработки коллекций (сортировки, поиска) не добавлены в сами классы коллекций, а используется Collections.sort и т.п. методы, поэтому GS-collections предложили идею «богатых» коллекций (rich collections), которые хранят в себе все методы обработки, поиска, сортировки, то есть вместо Collections.sort(list) вызывается просто list.sort. Поэтому библиотека предлагает свои аналоги стандартных коллекций и дополнительно ряд новых коллекций.

Официальная информация: [документация](#), [исходные коды](#), [документация пользователя](#), [javadoc](#).

Как подключить к проекту:

▼ [Maven](#), [Gradle](#), [Ivy](#)

Maven

```

<dependency>
  <groupId>com.goldmansachs</groupId>
  <artifactId>gs-collections-api</artifactId>
  <version>6.2.0</version>
</dependency>

<dependency>
  <groupId>com.goldmansachs</groupId>
  <artifactId>gs-collections</artifactId>
  <version>6.2.0</version>
</dependency>

<dependency>
  <groupId>com.goldmansachs</groupId>
  <artifactId>gs-collections-testutils</artifactId>
  <version>6.2.0</version>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>com.goldmansachs</groupId>

```

```
<artifactId>gs-collections-forkjoin</artifactId>
<version>6.2.0</version>
</dependency>

Gradle

compile 'com.goldmansachs:gs-collections-api:6.2.0'
compile 'com.goldmansachs:gs-collections:6.2.0'
testCompile 'com.goldmansachs:gs-collections-testutils:6.2.0'
compile 'com.goldmansachs:gs-collections-forkjoin:6.2.0'

Ivy

<dependency org="com.goldmansachs" name="gs-collections-api" rev="6.2.0" />
<dependency org="com.goldmansachs" name="gs-collections" rev="6.2.0" />
<dependency org="com.goldmansachs" name="gs-collections-testutils" rev="6.2.0" />
<dependency org="com.goldmansachs" name="gs-collections-forkjoin" rev="6.2.0"/>
```

Название	Аналог JDK	Описание
FastList	ArrayList	Аналог ArrayList с возможностью использовать функции вроде sort, select и т.п. прямо у объекта коллекции
UnifiedSet	HashSet	Аналог HashSet. См FastList
TreeSortedSet	TreeSet	Аналог TreeSet. См FastList
UnifiedMap	HashMap	Аналог HashMap. См FastList
TreeSortedMap	TreeMap	Аналог TreeMap. См FastList
HashBiMap	-	Реализация BiMap, см. Guava
HashBag	-	Реализация Multiset, см. Guava
TreeBag	-	Реализация отсортированного BiMap, см. Guava
ArrayStack	ArrayDeque	Реализация стека с порядком «last-in, first-out», похожего на класс Stack JDK
FastListMultimap	-	Реализация Multimap, см. Guava
IntArrayList, FloatHashSet, ArrayStack, HashBag, ByteIntHashMap	-	Коллекции примитивных различных типов, принцип наименования такой же как у trove, но кроме аналогов JDK, так же существуют аналоги коллекций Stack, Bag

3.5 Fastutil коллекции

Давайте очень кратко рассмотрим эту библиотеку для работы с коллекциями примитивных типов. Подробнее можно найти информацию: [документация](#), [исходные коды](#), [javadoc](#)

Название	Описание
Byte2DoubleOpenHashMap, IntArrayList, IntArrayPriorityQueue и т.п.	Коллекции различных примитивных типов, принцип наименования [Тип]ArrayList, [Тип]ArrayPriorityQueue и т.п. для списков или множеств, и [ТипКлюча]2[ТипЗначения]OpenHashMap и т.п. для Map.
IntBigList, DoubleOpenHashSet и т.п.	Коллекции различных примитивных типов очень Большого размера, эти коллекции позволяют использовать long элементов, вместо int. Внутри данные, как правило, хранятся как массивы массивов. Не рекомендуется использовать подобные коллекции там где хватит обычных, так как потери производительности могут достигать примерно 30%, однако такие коллекции позволяют работать с действительно большим количеством данных

3.6 Прочие библиотеки коллекций и немного о производительности примитивных коллекций

Кроме Trove и Fastutil есть ещё несколько известных библиотек, реализующих коллекции примитивных типов и более быстрые

аналоги стандартных коллекций:

- 1) **HPPC** — High Performance Primitive Collections for Java, так же предоставляет примитивные коллекции аналогичные коллекциям из JDK,
- 2) **Koloboke** (другое имя HFTC) — как можно понять из имени эту библиотеку примитивных типов разработал русский программист (Roman Leventov) в рамках проекта [OpenHFT](#). Библиотека так же служит для реализации высокопроизводительных примитивных коллекций.

Если интересно сравнение производительности разных библиотек советую посмотреть эту [статью](#), только нужно учитывать, что тестировали только коллекции HashMap и в определенных условиях. К тому же, замеряли только скорость работы, не учитывая занимаемую память (например, HashMap jdk могут занимать намного больше памяти чем аналоги от trove), а иногда память может быть даже более важной чем производительность.

Update: В комментариях, к статье было высказано мнение что Trove плохо использовать в новых проектах, так как он по всех параметрам уступает fastutil или GS (кол-во багов, полнота покрытия интерфейсов, производительность, активность поддержки, и т. д.). К сожалению, у меня нет возможности сейчас провести полноценный анализ/сравнение Trove с fastutil и GS, поэтому не могу проверить данное мнение, просто учитывайте его при выборе библиотеки альтернативных коллекций.

IV. Сравнение реализации самых популярных альтернативных коллекций в разных библиотеках

4.1 Реализация мультимножества (MultiSet/Bag) в библиотеках guava, Apache Commons Collections и GS

Итак, мультимножество это множество, которое сохраняет не только факт наличие элементов в множестве, но и количество вхождений в него. В JDK его можно эмулировать конструкцией Map <T, Integer>, но, естественно, специализированные коллекции позволяют использовать значительно меньше кода. Сравним какие реализации данной коллекции предлагают разные библиотеки:

Внимание: если таблица не помещается целиком, попробуйте уменьшить масштаб страницы или открыть в другом браузере.

Тип коллекции	Guava	Apache Commons Collections	GS Collections	JDK
Порядок коллекции не определен	HashMultiset	HashBag	HashBag	HashMap<String, Integer>
Отсортированная в заданном или натуральном порядке	TreeMultiset	TreeBag	TreeBag	TreeMap<String, Integer>
В порядке добавления	LinkedHashMultiset	-	-	LinkedHashMap<String, Integer>
Многопоточные	ConcurrentHashMultiset	SynchronizedBag	SynchronizedBag	Collections.synchronizedMap(HashMap<String, Integer>)
Многопоточные и отсортированные	-	SynchronizedSortedBag	SynchronizedSortedBag	Collections.synchronizedSortedMap(TreeMap<String, Integer>)
Не изменяемые	ImmutableMultiset	UnmodifiableBag	UnmodifiableBag	Collections.unmodifiableMap(HashMap<String, Integer>)
Не изменяемые и отсортированные	ImmutableSortedMultiset	UnmodifiableSortedBag	UnmodifiableSortedBag	Collections.unmodifiableSortedMap(TreeMap<String, Integer>)

Примеры использования мультимножества (MultiSet/Bag) для подсчета слов в тексте

Есть задача: дана строка текста «Hello World! Hello All! Hi World!», нужно разобрать её на отдельные слова где разделитель только пробел, сохранить в какую-нибудь коллекцию и вывести количество вхождений каждого слова, общее количество слов в тексте и количество уникальных слов.

Посмотрим как это сделать с помощью

1. разных вариантов Multiset от Guava:

▼ [Используем HashMultiset от guava для подсчета слов](#)

Обратите внимание, что порядок вывода в System.out.println(multiset) и в System.out.println(multiset.elementSet()) — произвольный, то есть не определен.

```
// Разберем текст на слова
String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Создаем Multiset
```

```

Multiset<String> multiset = HashMultiset.create(Arrays.asList(INPUT_TEXT.split(" ")));

// Выводим кол-вом вхождений слов
System.out.println(multiset); // напечатает [Hi, Hello x 2, World! x 2, All!] - в произвольном порядке
// Выводим все уникальные слова
System.out.println(multiset.elementSet()); // напечатает [Hi, Hello, World!, All!] - в произвольном
порядке

// Выводим количество по каждому слову
System.out.println("Hello = " + multiset.count("Hello")); // напечатает 2
System.out.println("World = " + multiset.count("World!")); // напечатает 2
System.out.println("All = " + multiset.count("All!")); // напечатает 1
System.out.println("Hi = " + multiset.count("Hi")); // напечатает 1
System.out.println("Empty = " + multiset.count("Empty")); // напечатает 0

// Выводим общее количества всех слов в тексте
System.out.println(multiset.size()); //напечатает 6

// Выводим общее количество всех уникальных слов
System.out.println(multiset.elementSet().size()); //напечатает 4

```

▼ [Используем TreeMultiset от guava для подсчета слов](#)

Обратите внимание, что порядок вывода в `System.out.println(multiset)` и в `System.out.println(multiset.elementSet())` — натуральный, то есть слова отсортированы по алфавиту.

```

// Разберем текст на слова
String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Создаем Multiset
Multiset<String> multiset = TreeMultiset.create(Arrays.asList(INPUT_TEXT.split(" ")));

// Выводим кол-вом вхождений слов
System.out.println(multiset); // напечатает [All!, Hello x 2, Hi, World! x 2]- в алфавитном порядке
// Выводим все уникальные слова
System.out.println(multiset.elementSet()); // напечатает [All!, Hello, Hi, World!]- в алфавитном порядке

// Выводим количество по каждому слову
System.out.println("Hello = " + multiset.count("Hello")); // напечатает 2
System.out.println("World = " + multiset.count("World!")); // напечатает 2
System.out.println("All = " + multiset.count("All!")); // напечатает 1
System.out.println("Hi = " + multiset.count("Hi")); // напечатает 1
System.out.println("Empty = " + multiset.count("Empty")); // напечатает 0

// Выводим общее количества всех слов в тексте
System.out.println(multiset.size()); //напечатает 6

// Выводим общее количество всех уникальных слов
System.out.println(multiset.elementSet().size()); //напечатает 4

```

▼ [Используем LinkedHashMapMultisetTest от guava для подсчета слов](#)

Обратите внимание, что порядок вывода в `System.out.println(multiset)` и в `System.out.println(multiset.elementSet())` — в порядке первого добавления элемента

```

// Разберем текст на слова
String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Создаем Multiset
Multiset<String> multiset = LinkedHashMapMultiset.create(Arrays.asList(INPUT_TEXT.split(" ")));

// Выводим кол-вом вхождений слов
System.out.println(multiset); // напечатает [Hello x 2, World! x 2, All!, Hi]- в порядке первого

```

```

добавления элемента
// Выводим все уникальные слова
System.out.println(multiset.elementSet()); // напечатает [Hello, World!, All!, Hi] - в порядке первого
добавления элемента

// Выводим количество по каждому слову
System.out.println("Hello = " + multiset.count("Hello")); // напечатает 2
System.out.println("World = " + multiset.count("World!")); // напечатает 2
System.out.println("All = " + multiset.count("All!")); // напечатает 1
System.out.println("Hi = " + multiset.count("Hi")); // напечатает 1
System.out.println("Empty = " + multiset.count("Empty")); // напечатает 0

// Выводим общее количество всех слов в тексте
System.out.println(multiset.size()); //напечатает 6

// Выводим общее количество всех уникальных слов
System.out.println(multiset.elementSet().size()); //напечатает 4

```

▼ Используем ConcurrentHashMultiset от guava для подсчета слов

Обратите внимание, что порядок вывода в `System.out.println(multiset)` и в `System.out.println(multiset.elementSet())` — произвольный, то есть не определен, так как это по сути многопоточная версия `HashMultiset`

```

// Разберем текст на слова
String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Создаем Multiset
Multiset<String> multiset = ConcurrentHashMultiset.create(Arrays.asList(INPUT_TEXT.split(" ")));

// Выводим кол-вом вхождений слов
System.out.println(multiset); // напечатает [Hi, Hello x 2, World! x 2, All!] - в произвольном порядке
// Выводим все уникальные слова
System.out.println(multiset.elementSet()); // напечатает [Hi, Hello, World!, All!] - в произвольном
порядке

// Выводим количество по каждому слову
System.out.println("Hello = " + multiset.count("Hello")); // напечатает 2
System.out.println("World = " + multiset.count("World!")); // напечатает 2
System.out.println("All = " + multiset.count("All!")); // напечатает 1
System.out.println("Hi = " + multiset.count("Hi")); // напечатает 1
System.out.println("Empty = " + multiset.count("Empty")); // напечатает 0

// Выводим общее количество всех слов в тексте
System.out.println(multiset.size()); //напечатает 6

// Выводим общее количество всех уникальных слов
System.out.println(multiset.elementSet().size()); //напечатает 4

```

2. разных вариантов Bag от Apache Commons Collections:

▼ Использование HashBag из Apache Commons Collections

Обратите внимание, что порядок вывода в `System.out.println(multiset)` и в `System.out.println(multiset.elementSet())` — произвольный, то есть не определен.

```

// Разберем текст на слова
String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Создаем Multiset
Bag bag = new HashBag(Arrays.asList(INPUT_TEXT.split(" ")));

// Выводим кол-вом вхождений слов
System.out.println(bag); // напечатает [1:Hi,2:Hello,2:World!,1:All!] - в произвольном порядке

```

```
// Выводим все уникальные слова
System.out.println(bag.uniqueSet()); // напечатает [Hi, Hello, World!, All!] - в произвольном порядке

// Выводим количество по каждому слову
System.out.println("Hello = " + bag.getCount("Hello")); // напечатает 2
System.out.println("World = " + bag.getCount("World!")); // напечатает 2
System.out.println("All = " + bag.getCount("All!")); // напечатает 1
System.out.println("Hi = " + bag.getCount("Hi")); // напечатает 1
System.out.println("Empty = " + bag.getCount("Empty")); // напечатает 0

// Выводим общее количества всех слов в тексте
System.out.println(bag.size()); //напечатает 6

// Выводим общее количество всех уникальных слов
System.out.println(bag.uniqueSet().size()); //напечатает 4
```

▼ [Использование TreeBag из Apache Commons Collections](#)

Обратите внимание, что порядок вывода в `System.out.println(multiset)` и в `System.out.println(multiset.elementSet())` — натуральный, то есть слова отсортированы по алфавиту.

```
// Разберем текст на слова
String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Создаем Multiset
Bag bag = new TreeBag(Arrays.asList(INPUT_TEXT.split(" ")));

// Выводим кол-вом вхождений слов
System.out.println(bag); // напечатает [1:All!,2:Hello,1:Hi,2:World!]- в алфавитном порядке
// Выводим все уникальные слова
System.out.println(bag.uniqueSet()); // напечатает [All!, Hello, Hi, World!]- в алфавитном порядке

// Выводим количество по каждому слову
System.out.println("Hello = " + bag.getCount("Hello")); // напечатает 2
System.out.println("World = " + bag.getCount("World!")); // напечатает 2
System.out.println("All = " + bag.getCount("All!")); // напечатает 1
System.out.println("Hi = " + bag.getCount("Hi")); // напечатает 1
System.out.println("Empty = " + bag.getCount("Empty")); // напечатает 0

// Выводим общее количества всех слов в тексте
System.out.println(bag.size()); //напечатает 6

// Выводим общее количество всех уникальных слов
System.out.println(bag.uniqueSet().size()); //напечатает 4
```

▼ [Использование SynchronizedBag из Apache Commons Collections](#)

Обратите внимание, что порядок вывода в `System.out.println(multiset)` и в `System.out.println(multiset.elementSet())` — произвольный, то есть не определен, так как это по сути многопоточная версия HashBag

```
// Разберем текст на слова
String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Создаем Multiset
Bag bag = SynchronizedBag.synchronizedBag(new HashBag(Arrays.asList(INPUT_TEXT.split(" "))));

// Выводим кол-вом вхождений слов
System.out.println(bag); // напечатает [1:Hi,2:Hello,2:World!,1:All!] - в произвольном порядке
// Выводим все уникальные слова
System.out.println(bag.uniqueSet()); // напечатает [Hi, Hello, World!, All!] - в произвольном порядке

// Выводим количество по каждому слову
```

```

System.out.println("Hello = " + bag.getCount("Hello")); // напечатает 2
System.out.println("World = " + bag.getCount("World!")); // напечатает 2
System.out.println("All = " + bag.getCount("All!")); // напечатает 1
System.out.println("Hi = " + bag.getCount("Hi")); // напечатает 1
System.out.println("Empty = " + bag.getCount("Empty")); // напечатает 0

// Выводим общее количества всех слов в тексте
System.out.println(bag.size()); //напечатает 6

// Выводим общее количество всех уникальных слов
System.out.println(bag.uniqueSet().size()); //напечатает 4

```

▼ Использование SynchronizedSortedBag из Apache Commons Collections

Обратите внимание, что порядок вывода в `System.out.println(multiset)` и в `System.out.println(multiset.elementSet())` — натуральный, то есть слова отсортированы по алфавиту, так как это по сути многопоточная версия `SortedBag`

```

// Разберем текст на слова
String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Создаем Multiset
Bag bag = SynchronizedSortedBag.synchronizedBag(new TreeBag(Arrays.asList(INPUT_TEXT.split(" "))));

// Выводим кол-вом вхождений слов
System.out.println(bag); // напечатает [1:All!,2:Hello,1:Hi,2:World!]- в алфавитном порядке
// Выводим все уникальные слова
System.out.println(bag.uniqueSet()); // напечатает [All!, Hello, Hi, World!]- в алфавитном порядке

// Выводим количество по каждому слову
System.out.println("Hello = " + bag.getCount("Hello")); // напечатает 2
System.out.println("World = " + bag.getCount("World!")); // напечатает 2
System.out.println("All = " + bag.getCount("All!")); // напечатает 1
System.out.println("Hi = " + bag.getCount("Hi")); // напечатает 1
System.out.println("Empty = " + bag.getCount("Empty")); // напечатает 0

// Выводим общее количества всех слов в тексте
System.out.println(bag.size()); // напечатает 6

// Выводим общее количество всех уникальных слов
System.out.println(bag.uniqueSet().size()); // напечатает 4

```

3. разных вариантов Bag от GS Collections:

▼ Использование MutableBag из GS Collections

Обратите внимание, что порядок вывода в `System.out.println(bag)` и в `System.out.println(bag.toSet())` — не определен

```

// Разберем текст на слова
String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Создаем Multiset
MutableBag<String> bag = HashBag.newBag(Arrays.asList(INPUT_TEXT.split(" ")));

// Выводим кол-вом вхождений слов
System.out.println(bag); // напечатает [Hi, World!, World!, Hello, Hello, All!]- в произвольном порядке
// Выводим все уникальные слова
System.out.println(bag.toSet()); // напечатает [Hi, Hello, World!, All!] - в произвольном порядке

// Выводим количество по каждому слову
System.out.println("Hello = " + bag.occurrencesOf("Hello")); // напечатает 2
System.out.println("World = " + bag.occurrencesOf("World!")); // напечатает 2

```

```

System.out.println("All = " + bag.occurrencesOf("All!")); // напечатает 1
System.out.println("Hi = " + bag.occurrencesOf("Hi")); // напечатает 1
System.out.println("Empty = " + bag.occurrencesOf("Empty")); // напечатает 0

// Выводим общее количества всех слов в тексте
System.out.println(bag.size()); //напечатает 6

// Выводим общее количество всех уникальных слов
System.out.println(bag.toSet().size()); //напечатает 4

```

▼ [Использование MutableSortedBag из GS Collections](#)

Обратите внимание, что порядок вывода в `System.out.println(bag)` и в `System.out.println(bag.toSortedSet())` — будет натуральным, т.е. по алфавиту в данном случае

```

// Разберем текст на слова
String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Создаем Multiset
MutableSortedBag<String> bag = TreeBag.newBag(Arrays.asList(INPUT_TEXT.split(" ")));

// Выводим кол-вом вхождений слов
System.out.println(bag); // напечатает [All!, Hello, Hello, Hi, World!, World!]- в натуральном порядке
// Выводим все уникальные слова
System.out.println(bag.toSortedSet()); // напечатает [All!, Hello, Hi, World!]- в натуральном порядке

// Выводим количество по каждому слову
System.out.println("Hello = " + bag.occurrencesOf("Hello")); // напечатает 2
System.out.println("World = " + bag.occurrencesOf("World!")); // напечатает 2
System.out.println("All = " + bag.occurrencesOf("All!")); // напечатает 1
System.out.println("Hi = " + bag.occurrencesOf("Hi")); // напечатает 1
System.out.println("Empty = " + bag.occurrencesOf("Empty")); // напечатает 0

// Выводим общее количества всех слов в тексте
System.out.println(bag.size()); //напечатает 6

// Выводим общее количество всех уникальных слов
System.out.println(bag.toSet().size()); //напечатает 4

```

4. Ну и наконец, посмотрим как можно сделать тоже самое в чистом JDK с помощью эмуляции multiSet через HashMap

▼ [Эмуляция multiSet через HashMap](#)

Как вы легко можете заметить, кода потребовалось, естественно, больше чем у любых реализаций multiSet или Bag.

```

// Разберем текст на слова
String INPUT_TEXT = "Hello World! Hello All! Hi World!";
List<String> listResult = Arrays.asList(INPUT_TEXT.split(" "));
// Создаем эмуляцию Multiset с помощью HashMap и заполняем
Map<String, Integer> fakeMultiset = new HashMap<String,Integer>(listResult.size());

for(String word: listResult) {
    Integer cnt = fakeMultiset.get(word);
    fakeMultiset.put(word, cnt == null ? 1 : cnt + 1);
}

// Выводим кол-вом вхождений слов
System.out.println(fakeMultiset); // напечатает {World!=2, Hi=1, Hello=2, All!=1}- в произвольном порядке
// Выводим все уникальные слова
System.out.println(fakeMultiset.keySet()); // напечатает [World!, Hi, Hello, All!] - в произвольном порядке

```



```
// Выводим количество по каждому слову
System.out.println("Hello = " + fakeMultiset.get("Hello")); // напечатает 2
System.out.println("World = " + fakeMultiset.get("World!")); // напечатает 2
System.out.println("All = " + fakeMultiset.get("All!")); // напечатает 1
System.out.println("Hi = " + fakeMultiset.get("Hi")); // напечатает 1
System.out.println("Empty = " + fakeMultiset.get("Empty")); // напечатает null

// Выводим общее количества всех слов в тексте
Integer cnt = 0;
for (Integer wordCount : fakeMultiset.values()){
    cnt += wordCount;
}
System.out.println(cnt); //напечатает 6

// Выводим общее количество уникальных слов
System.out.println(fakeMultiset.size()); //напечатает 4
```

4.2 Реализация Multimap в библиотеках guava, Apache Commons Collections и GS Collections

Итак, Multimap это map, у которой у каждого ключа есть набор значений. Давайте сравним какие реализации есть данной коллекции в разных библиотеках. В таблице ниже порядок ключей и порядок значений показывает как будет происходить итерирования по ключам и значениям соответственно, дубликаты — может ли коллекция значений содержать дубликаты, аналог ключей и значений — на каких коллекциях построены ключи и значения, JDK показывает аналог коллекции с помощью JDK коллекций.

Внимание: если таблица не помещается целиком, попробуйте уменьшить масштаб страницы или открыть в другом браузере.

Порядо к ключей	Порядок значени й	Дуб- лика - ты	Аналог ключей	Аналог значе- ний	Guava	Apache Commons Collections	GS Collections	JDK
не задан	в порядке добавле- ния	да	HashMap	Arraylis t	ArrayList- Multimap	MultiValueMap	FastList- Multimap	HashMap<K, ArrayList<V>>
не задан	не задан	нет	HashMap	HashSet	HashMultima p	MultiValueMap. multiValueMap(new HashMap<K, Set>(), HashSet.class);	UnifiedSet- Multimap	HashMap<K, HashSet<V>>
не задан	отсорти- рован	нет	HashMap	TreeSet	Multimaps. newMultimap (HashMap, Supplier <TreeSet>)	MultiValueMap. multiValueMap(new HashMap<K, Set>(), TreeSet.class);	TreeSortedSet- Multimap	HashMap<K, TreeSet<V>>
в порядке добавле- ния	в порядке добавле- ния	да	Linked HashMap	Arraylis t	LinkedList- Multimap	MultiValueMap. multiValueMap(new LinkedHashMap<K, List>(), ArrayList.class);		LinkedHashMap< K, ArrayList<V>>
в порядке добавле- ния	в порядке добавле- ния	нет	LinkedHash - Multimap	Linked- HashSet	LinkedHash- Multimap	MultiValueMap. multiValueMap(new LinkedHashMap<K, Set> (, LinkedHashSet.class);		LinkedHashMap<K, LinkedHashSet<V> >
отсорти- рован	отсорти- рован	нет	TreeMap	TreeSet	TreeMultimap	MultiValueMap. multiValueMap(new TreeMap<K, Set>(),TreeSet.class);		TreeMap<K, TreeSet<V>>

Как видно из таблицы, в Apache Commons Collections есть лишь одна реализация данного вида коллекции, остальные можно получить оборачивая стандартные коллекции при создании. В guava намного больше уже определенных коллекций, при этом есть возможность реализовать обертку над любой map'ами и любыми коллекциями значений. В GS Collections есть так же

специальные коллекция multimap основанная на Bag (HashBagMultimap), см. multiset и multimap.

Примеры использования Multimap для сохранения всех вхождений слов в тексте

Есть задача: дана строка текста «Hello World! Hello All! Hi World!», нужно разобрать её на отдельные слова где разделитель только пробел и теперь нам нужно знать не только сколько каждых слов в тексте, но и все индекс вхождения слова в тексте, то есть что Hello это первое и третье слово в тексте и т.д.

Посмотрим как это сделать с помощью

1. разных вариантов Multimap от Guava:

▼ [Используем HashMultimap от guava](#)

Обратите внимание, что порядок хранения данных произвольный как для ключей, так и для значений (для ключей поведение аналогичное HashMap, для значений HashSet). Повторяющиеся значения для одного ключа игнорируются.

```
String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Разберем текст на слова и индексы

List
<String> words = Arrays.asList(INPUT_TEXT.split(" "));

// Создаем Multimap
Multimap<String, Integer> multiMap = HashMultimap.create();

// Заполним Multimap
int i = 0;
for(String word: words) {
    multiMap.put(word, i);
    i++;
}

// Выводим все вхождения слов в текст
System.out.println(multiMap); // напечатает {Hi=[4], Hello=[0, 2], World!=[1, 5], All!=[3]} - в
произвольном порядке
// Выводим все уникальные слова
System.out.println(multiMap.keySet()); // напечатает [Hi, Hello, World!, All!] - в произвольном порядке

// Выводим все индексы вхождения слова в текст
System.out.println("Hello = " + multiMap.get("Hello")); // напечатает [0, 2]
System.out.println("World = " + multiMap.get("World!")); // напечатает [1, 5]
System.out.println("All = " + multiMap.get("All!")); // напечатает [3]
System.out.println("Hi = " + multiMap.get("Hi")); // напечатает [4]
System.out.println("Empty = " + multiMap.get("Empty")); // напечатает []

// Выводим общее количества всех слов в тексте
System.out.println(multiMap.size()); //напечатает 6

// Выводим общее количество всех уникальных слов
System.out.println(multiMap.keySet().size()); //напечатает 4
```

▼ [Используем ArrayListMultimapTest от guava](#)

Обратите внимание, что порядок хранения данных произвольный для ключей, так и в порядке добавления для значений (для ключей поведение аналогичное HashMap, для значений ArrayList). Повторяющиеся значения для одного ключа сохраняются.

```
String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Разберем текст на слова и индексы

List
<String> words = Arrays.asList(INPUT_TEXT.split(" "));

// Создаем Multimap
Multimap<String, Integer> multiMap = ArrayListMultimap.create();

// Заполним Multimap
int i = 0;
for(String word: words) {
```

```

multiMap.put(word, i);
i++;
}

// Выводим все вхождения слов в текст
System.out.println(multiMap); // напечатает {Hi=[4], Hello=[0, 2], World!=[1, 5], All!=[3]} - ключи в
произвольном порядке, значения в порядке добавления
// Выводим все уникальные слова
System.out.println(multiMap.keySet()); // напечатает [Hello, World!, All!, Hi]- в произвольном порядке

// Выводим все индексы вхождения слова в текст
System.out.println("Hello = " + multiMap.get("Hello")); // напечатает [0, 2]
System.out.println("World = " + multiMap.get("World!")); // напечатает [1, 5]
System.out.println("All = " + multiMap.get("All!")); // напечатает [3]
System.out.println("Hi = " + multiMap.get("Hi")); // напечатает [4]
System.out.println("Empty = " + multiMap.get("Empty")); // напечатает []

// Выводим общее количества всех слов в тексте
System.out.println(multiMap.size()); //напечатает 6

// Выводим общее количество всех уникальных слов
System.out.println(multiMap.keySet().size()); //напечатает 4

```

▼ [Используем LinkedHashMapMultimapTest or guava](#)

Обратите внимание, что порядок хранения данных в порядке добавления для ключей и для значений (для ключей поведение аналогичное LinkedHashMap, для значений LinkedHashSet). Повторяющиеся значения для одного ключа игнорируются.

```

String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Разберем текст на слова и индексы
List
<String> words = Arrays.asList(INPUT_TEXT.split(" "));
// Создаем Multimap
Multimap<String, Integer> multiMap = LinkedHashMapMultimap.create();

// Заполним Multimap
int i = 0;
for(String word: words) {
multiMap.put(word, i);
i++;
}

// Выводим все вхождения слов в текст
System.out.println(multiMap); // напечатает {Hello=[0, 2], World!=[1, 5], All!=[3], Hi=[4]}-в порядке
добавления
// Выводим все уникальные слова
System.out.println(multiMap.keySet()); // напечатает [Hello, World!, All!, Hi]- в порядке добавления

// Выводим все индексы вхождения слова в текст
System.out.println("Hello = " + multiMap.get("Hello")); // напечатает [0, 2]
System.out.println("World = " + multiMap.get("World!")); // напечатает [1, 5]
System.out.println("All = " + multiMap.get("All!")); // напечатает [3]
System.out.println("Hi = " + multiMap.get("Hi")); // напечатает [4]
System.out.println("Empty = " + multiMap.get("Empty")); // напечатает []

// Выводим общее количества всех слов в тексте
System.out.println(multiMap.size()); //напечатает 6

// Выводим общее количество всех уникальных слов
System.out.println(multiMap.keySet().size()); //напечатает 4

```

▼ Используем `LinkedListMultimapTest` от `guava`

Обратите внимание, что порядок хранения данных в порядке добавления для ключей и для значений (для ключей поведение аналогичное `LinkedHashMap`, для значений `LinkedList`). Повторяющиеся значения для одного ключа сохраняются.

```
String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Разберем текст на слова и индексы
List
<String> words = Arrays.asList(INPUT_TEXT.split(" "));

// Создаем Multimap
Multimap<String, Integer> multiMap = LinkedListMultimap.create();

// Заполним Multimap
int i = 0;
for(String word: words) {
    multiMap.put(word, i);
    i++;
}

// Выводим все вхождения слов в текст
System.out.println(multiMap); // напечатает {Hello=[0, 2], World!=[1, 5], All!=[3], Hi=[4]}-в порядке
добавления
// Выводим все уникальные слова
System.out.println(multiMap.keySet()); // напечатает [Hello, World!, All!, Hi]- в порядке добавления

// Выводим все индексы вхождения слова в текст
System.out.println("Hello = " + multiMap.get("Hello")); // напечатает [0, 2]
System.out.println("World = " + multiMap.get("World!")); // напечатает [1, 5]
System.out.println("All = " + multiMap.get("All!")); // напечатает [3]
System.out.println("Hi = " + multiMap.get("Hi")); // напечатает [4]
System.out.println("Empty = " + multiMap.get("Empty")); // напечатает []

// Выводим общее количества всех слов в тексте
System.out.println(multiMap.size()); //напечатает 6

// Выводим общее количество всех уникальных слов
System.out.println(multiMap.keySet().size()); //напечатает 4
```

▼ Используем `TreeMultimapTest` от `guava`

Обратите внимание, что порядок хранения данных отсортированный для ключей и для значений (для ключей поведение аналогичное `TreeMap`, для значений `TreeSet`).

```
String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Разберем текст на слова и индексы
List<String> words = Arrays.asList(INPUT_TEXT.split(" "));

// Создаем Multimap
Multimap<String, Integer> multiMap = TreeMultimap.create();

// Заполним Multimap
int i = 0;
for(String word: words) {
    multiMap.put(word, i);
    i++;
}

// Выводим все вхождения слов в текст
System.out.println(multiMap); // напечатает {Hello=[0, 2], World!=[1, 5], All!=[3], Hi=[4]}-в
натуральном порядке
// Выводим все уникальные слова
System.out.println(multiMap.keySet()); // напечатает [Hello, World!, All!, Hi]- в натуральном порядке
```

```
// Выводим все индексы вхождения слова в текст
System.out.println("Hello = " + multiMap.get("Hello")); // напечатает [0, 2]
System.out.println("World = " + multiMap.get("World!")); // напечатает [1, 5]
System.out.println("All = " + multiMap.get("All!")); // напечатает [3]
System.out.println("Hi = " + multiMap.get("Hi")); // напечатает [4]
System.out.println("Empty = " + multiMap.get("Empty")); // напечатает []

// Выводим общее количества всех слов в тексте
System.out.println(multiMap.size()); //напечатает 6

// Выводим общее количество всех уникальных слов
System.out.println(multiMap.keySet().size()); //напечатает 4
```

2. разных вариантов MultiValueMap от Apache Commons Collections:

▼ Используем MultiValueMap от Apache Commons Collections

Обратите внимание, что порядок хранения данных произвольный для ключей и значений (используется HashMap для ключей и ArrayList для значений)

```
String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Разберем текст на слова и индексы
List<String> words = Arrays.asList(INPUT_TEXT.split(" "));
// Создаем Multimap
MultiMap<String, Integer> multiMap = new MultiValueMap<String, Integer>();

// Заполним Multimap
int i = 0;
for(String word: words) {
    multiMap.put(word, i);
    i++;
}

// Выводим все вхождения слов в текст
System.out.println(multiMap); // напечатает {Hi=[4], Hello=[0, 2], World!=[1, 5], All!=[3]} - в
произвольном порядке
// Выводим все уникальные слова
System.out.println(multiMap.keySet()); // напечатает [Hi, Hello, World!, All!] - в произвольном порядке

// Выводим все индексы вхождения слова в текст
System.out.println("Hello = " + multiMap.get("Hello")); // напечатает [0, 2]
System.out.println("World = " + multiMap.get("World!")); // напечатает [1, 5]
System.out.println("All = " + multiMap.get("All!")); // напечатает [3]
System.out.println("Hi = " + multiMap.get("Hi")); // напечатает [4]
System.out.println("Empty = " + multiMap.get("Empty")); // напечатает null

// Выводим общее количество всех уникальных слов
System.out.println(multiMap.keySet().size()); //напечатает 4
```

▼ Используем MultiValueMap, оборачивающий TreeMap<String, TreeSet>()

Обратите внимание, что порядок хранения данных отсортированный для ключей и значений

```
String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Разберем текст на слова и индексы
List<String> words = Arrays.asList(INPUT_TEXT.split(" "));
// Создаем Multimap
MultiMap<String, Integer> multiMap = MultiValueMap.multiValueMap(new TreeMap<String, Set>(), TreeSet.class);

// Заполним Multimap
```

```

int i = 0;
for(String word: words) {
    multiMap.put(word, i);
    i++;
}

// Выводим все вхождения слов в текст
System.out.println(multiMap); // напечатает {All!= [3], Hello=[0, 2], Hi=[4], World!= [1, 5]} -в
натуральном порядке
// Выводим все уникальные слова
System.out.println(multiMap.keySet()); // напечатает [All!, Hello, Hi, World!] в натуральном порядке

// Выводим все индексы вхождения слова в текст
System.out.println("Hello = " + multiMap.get("Hello")); // напечатает [0, 2]
System.out.println("World = " + multiMap.get("World!")); // напечатает [1, 5]
System.out.println("All = " + multiMap.get("All!")); // напечатает [3]
System.out.println("Hi = " + multiMap.get("Hi")); // напечатает [4]
System.out.println("Empty = " + multiMap.get("Empty")); // напечатает null

// Выводим общее количество всех уникальных слов
System.out.println(multiMap.keySet().size()); //напечатает 4

```

▼ Используем MultiValueMap, оборачивающий LinkedHashMap<String, LinkedHashSet>()

Обратите внимание, что порядок хранения данных по первому добавлению для ключей и значений

```

String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Разберем текст на слова и индексы
List<String> words = Arrays.asList(INPUT_TEXT.split(" "));

// Создаем Multimap
MultiMap<String, Integer> multiMap = MultiValueMap.multiValueMap(new LinkedHashMap<String, Set>(), LinkedHashSet.c
lass);

// Заполним Multimap
int i = 0;
for(String word: words) {
    multiMap.put(word, i);
    i++;
}

// Выводим все вхождения слов в текст
System.out.println(multiMap); // напечатает {Hello=[0, 2], World!= [1, 5], All!= [3], Hi=[4]} - в порядке
добавления
// Выводим все уникальные слова
System.out.println(multiMap.keySet()); // напечатает [Hello, World!, All!, Hi] - в порядке добавления

// Выводим все индексы вхождения слова в текст
System.out.println("Hello = " + multiMap.get("Hello")); // напечатает [0, 2]
System.out.println("World = " + multiMap.get("World!")); // напечатает [1, 5]
System.out.println("All = " + multiMap.get("All!")); // напечатает [3]
System.out.println("Hi = " + multiMap.get("Hi")); // напечатает [4]
System.out.println("Empty = " + multiMap.get("Empty")); // напечатает null

// Выводим общее количество всех уникальных слов
System.out.println(multiMap.keySet().size()); //напечатает 4

```

3. разных вариантов Multimap от GS Collections:

▼ [Использование FastListMultimap](#)


```

String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Разберем текст на слова и индексы
List<String> words = Arrays.asList(INPUT_TEXT.split(" "));

// Создаем Multimap
MutableListMultimap<String, Integer> multiMap = new FastListMultimap<String, Integer>();

// Заполним Multimap
int i = 0;
for(String word: words) {
    multiMap.put(word, i);
    i++;
}

// Выводим все вхождения слов в текст
System.out.println(multiMap); // напечатает {Hi=[4], World!=[1, 5], Hello=[0, 2], All!=[3]}- в
произвольном порядке
// Выводим все уникальные слова
System.out.println(multiMap.keySet()); // напечатает [Hi, Hello, World!, All!] - в произвольном
порядке

// Выводим все индексы вхождения слова в текст
System.out.println("Hello = " + multiMap.get("Hello")); // напечатает [0, 2]
System.out.println("World = " + multiMap.get("World!")); // напечатает [1, 5]
System.out.println("All = " + multiMap.get("All!")); // напечатает [3]
System.out.println("Hi = " + multiMap.get("Hi")); // напечатает [4]
System.out.println("Empty = " + multiMap.get("Empty")); // напечатает []

// Выводим общее количество всех слов в тексте
System.out.println(multiMap.size()); //напечатает 6

// Выводим общее количество уникальных слов в тексте
System.out.println(multiMap.keySet().size()); //напечатает 4

```

▼ Использование HashBagMultimap

```

String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Разберем текст на слова и индексы
List<String> words = Arrays.asList(INPUT_TEXT.split(" "));

// Создаем Multimap
MutableBagMultimap<String, Integer> multiMap = new HashBagMultimap<String, Integer>();

// Заполним Multimap
int i = 0;
for(String word: words) {
    multiMap.put(word, i);
    i++;
}

// Выводим все вхождения слов в текст
System.out.println(multiMap); // напечатает {Hi=[4], World!=[1, 5], Hello=[0, 2], All!=[3]}- в
произвольном порядке
// Выводим все уникальные слова
System.out.println(multiMap.keySet()); // напечатает [Hi, Hello, World!, All!] - в произвольном
порядке

// Выводим все индексы вхождения слова в текст
System.out.println("Hello = " + multiMap.get("Hello")); // напечатает [0, 2]
System.out.println("World = " + multiMap.get("World!")); // напечатает [1, 5]
System.out.println("All = " + multiMap.get("All!")); // напечатает [3]

```

```

System.out.println("Hi = " + multiMap.get("Hi")); // напечатает [4]
System.out.println("Empty = " + multiMap.get("Empty")); // напечатает []

// Выводим общее количество всех слов в тексте
System.out.println(multiMap.size()); //напечатает 6

// Выводим общее количество уникальных слов в тексте
System.out.println(multiMap.keySet().size()); //напечатает 4

```

▼ [Использование TreeSortedSetMultimap](#)

```

String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Разберем текст на слова и индексы
List<String> words = Arrays.asList(INPUT_TEXT.split(" "));

// Создаем Multimap
MutableSortedSetMultimap<String, Integer> multiMap = new TreeSortedSetMultimap<String, Integer>();

// Заполним Multimap
int i = 0;
for(String word: words) {
    multiMap.put(word, i);
    i++;
}

// Выводим все вхождения слов в текст
System.out.println(multiMap); // напечатает {Hi=[4], World!=[1, 5], Hello=[0, 2], All!=[3]}- в
произвольном порядке
// Выводим все уникальные слова
System.out.println(multiMap.keySet()); // напечатает [Hi, Hello, World!, All!] - в произвольном
порядке

// Выводим все индексы вхождения слова в текст
System.out.println("Hello = " + multiMap.get("Hello")); // напечатает [0, 2]
System.out.println("World = " + multiMap.get("World!")); // напечатает [1, 5]
System.out.println("All = " + multiMap.get("All!")); // напечатает [3]
System.out.println("Hi = " + multiMap.get("Hi")); // напечатает [4]
System.out.println("Empty = " + multiMap.get("Empty")); // напечатает []

// Выводим общее количество всех слов в тексте
System.out.println(multiMap.size()); //напечатает 6

// Выводим общее количество уникальных слов в тексте
System.out.println(multiMap.keySet().size()); //напечатает 4

```

4. Ну и наконец, посмотрим как можно сделать тоже самое в чистом JDK с помощью эмуляции multiMap через HashMap

▼ [Эмуляция multiMap через HashMap](#)

```

final int LIST_INDEXES_CAPACITY = 50;
String INPUT_TEXT = "Hello World! Hello All! Hi World!";
// Разберем текст на слова и индексы
List<String> words = Arrays.asList(INPUT_TEXT.split(" "));

// Создаем Multimap
Map<String, List<Integer>> fakeMultiMap = new HashMap<String, List<Integer>>(words.size());

// Заполним map
int i = 0;

```

```

for(String word: words) {
    List<Integer> indexes = fakeMultiMap.get(word);
    if(indexes == null) {
        indexes = new ArrayList<Integer>(LIST_INDEXES_CAPACITY);
        fakeMultiMap.put(word, indexes);
    }
    indexes.add(i);
    i++;
}

// Выводим все вхождения слов в текст
System.out.println(fakeMultiMap); // напечатает {Hi=[4], Hello=[0, 2], World!=[1, 5],
All!=[3]} - в произвольном порядке
// Выводим все уникальные слова
System.out.println(fakeMultiMap.keySet()); // напечатает [Hi, Hello, World!, All!] - в
произвольном порядке

// Выводим все индексы вхождения слова в текст
System.out.println("Hello = " + fakeMultiMap.get("Hello")); // напечатает [0, 2]
System.out.println("World = " + fakeMultiMap.get("World!")); // напечатает [1, 5]
System.out.println("All = " + fakeMultiMap.get("All!")); // напечатает [3]
System.out.println("Hi = " + fakeMultiMap.get("Hi")); // напечатает [4]
System.out.println("Empty = " + fakeMultiMap.get("Empty")); // напечатает null

// Выводим общее количество всех слов в тексте
int cnt = 0;
for(List<Integer> lists: fakeMultiMap.values()) {
    cnt += lists.size();
}
System.out.println(cnt); //напечатает 6

// Выводим общее количество уникальных слов в тексте
System.out.println(fakeMultiMap.keySet().size()); //напечатает 4

```

4.3 Реализация BiMap в библиотеках guava, Apache Commons Collections и GS Collections

Реализация BiMap во всех библиотеках достаточно похожа, за исключением названия HashBiMap в guava и GS Collections и BidiMap в Apache Commons Collections. Кроме простейшей HashBiMap, у guava есть отдельные коллекции для работы с Enum в качестве ключей или значений, такие как EnumHashBiMap или EnumBiMap, у Apache Commons Collections есть ряд коллекций, где ключи упорядочены по добавлению или отсортированы.

Примеры использования BiMap для создания русско-английского «переводчика», действующего в обе стороны

Есть задача: есть массивы английских и русских слов соответствующие друг другу, нужно реализовать коллекцию русско-английского словаря, с возможностью перевода в обе стороны.

Посмотрим как это сделать с помощью

1. разных вариантов BiMap от Guava:

▼ [Используем BiMap от guava](#)

```

String[] englishWords = {"one", "two", "three", "ball", "snow"};
String[] russianWords = {"один", "два", "три", "мяч", "снег"};

// Создаем Multiset
BiMap<String, String> biMap = HashBiMap.create(englishWords.length);
// создаем англо-русский словарь
int i = 0;
for(String englishWord: englishWords) {
    biMap.put(englishWord, russianWords[i]);
}

```

```

i++;
}

// Выводим кол-вом вхождений слов
System.out.println(biMap); // напечатает {two=два, three=три, snow=снег, ball=мяч, one=один} - в произвольном порядке
// Выводим все уникальные слова
System.out.println(biMap.keySet()); // напечатает [two, three, snow, ball, one] - в произвольном порядке
System.out.println(biMap.values()); // напечатает [два, три, снег, мяч, один]- в произвольном порядке

// Выводим перевод по каждому слову
System.out.println("one = " + biMap.get("one")); // напечатает one = один
System.out.println("two = " + biMap.get("two")); // напечатает two = два
System.out.println("мяч = " + biMap.inverse().get("мяч")); // напечатает мяч = ball
System.out.println("снег = " + biMap.inverse().get("снег")); // напечатает снег = snow
System.out.println("empty = " + biMap.get("empty")); // напечатает empty = null

// Выводим общее количество переводов в словаре
System.out.println(biMap.size()); //напечатает 5

```

▼ Используем EnumBiMap от guava

```

enum ENGLISH_WORD {
    ONE, TWO, THREE, BALL, SNOW
}

enum POLISH_WORD {
    JEDEN, DWA, TRZY, KULA, SNIEG
}

// Задача даны массивы польско-английского перевода, сделать коллекцию для перевода слова в двух направлениях

public static void main(String[] args) {
    ENGLISH_WORD[] englishWords = ENGLISH_WORD.values();
    POLISH_WORD[] polishWords = POLISH_WORD.values();

    // Создаем Multiset
    BiMap<ENGLISH_WORD, POLISH_WORD> biMap = EnumBiMap.create(ENGLISH_WORD.class, POLISH_WORD.class);
    // создаем англо-польский словарь
    int i = 0;
    for(ENGLISH_WORD englishWord: englishWords) {
        biMap.put(englishWord, polishWords[i]);
        i++;
    }

    // Выводим кол-вом вхождений слов
    System.out.println(biMap); // напечатает {ONE=JEDEN, TWO=DWA, THREE=TRZY, BALL=KULA, SNOW=SNIEG}
    // Выводим все уникальные слова
    System.out.println(biMap.keySet()); // напечатает [ONE, TWO, THREE, BALL, SNOW]
    System.out.println(biMap.values()); // напечатает [JEDEN, DWA, TRZY, KULA, SNIEG]

    // Выводим перевод по каждому слову
    System.out.println("one = " + biMap.get(ENGLISH_WORD.ONE)); // напечатает one = JEDEN
    System.out.println("two = " + biMap.get(ENGLISH_WORD.TWO)); // напечатает two = DWA
    System.out.println("kula = " + biMap.inverse().get(POLISH_WORD.KULA)); // напечатает kula = BALL
    System.out.println("snieg = " + biMap.inverse().get(POLISH_WORD.SNIEG)); // напечатает snieg = SNOW
    System.out.println("empty = " + biMap.get("empty")); // напечатает empty = null

    // Выводим общее количество переводов в словаре
    System.out.println(biMap.size()); //напечатает 5

}

```

▼ [Используем EnumHashMap от guava](#)

```
enum ENGLISH_WORD {
    ONE, TWO, THREE, BALL, SNOW
}

// Задача даны массивы русско-английского перевода, сделать коллекцию для перевода слова в двух
// направлениях

public static void main(String[] args) {
    ENGLISH_WORD[] englishWords = ENGLISH_WORD.values();
    String[] russianWords = {"один", "два", "три", "мяч", "снег"};

    // Создаем Multiset
    BiMap<ENGLISH_WORD, String> biMap = EnumHashMap.create(ENGLISH_WORD.class);
    // создаем англо-русский словарь

    int i = 0;
    for(ENGLISH_WORD englishWord: englishWords) {
        biMap.put(englishWord, russianWords[i]);
        i++;
    }

    // Выводим кол-вом вхождений слов
    System.out.println(biMap); // напечатает {ONE=один, TWO=два, THREE=три, BALL=мяч, SNOW=снег}

    // Выводим все уникальные слова
    System.out.println(biMap.keySet()); // напечатает [ONE, TWO, THREE, BALL, SNOW]
    System.out.println(biMap.values()); // напечатает [один, два, три, мяч, снег]

    // Выводим перевод по каждому слову
    System.out.println("one = " + biMap.get(ENGLISH_WORD.ONE)); // напечатает one = один
    System.out.println("two = " + biMap.get(ENGLISH_WORD.TWO)); // напечатает two = два
    System.out.println("мяч = " + biMap.inverse().get("мяч")); // напечатает мяч = BALL
    System.out.println("снег = " + biMap.inverse().get("снег")); // напечатает снег = SNOW
    System.out.println("empty = " + biMap.get("empty")); // напечатает empty = null

    // Выводим общее количество переводов в словаре
    System.out.println(biMap.size()); //напечатает 5

}
```

2. С помощью BiDiMap от Apache Commons Collections:

▼ [Используем DualHashBiDiMap от Apache Commons Collections](#)

```
String[] englishWords = {"one", "two", "three", "ball", "snow"};
String[] russianWords = {"один", "два", "три", "мяч", "снег"};

// Создаем Multiset
BiDiMap<String, String> biMap = new DualHashBiDiMap();
// создаем англо-русский словарь

int i = 0;
for(String englishWord: englishWords) {
    biMap.put(englishWord, russianWords[i]);
    i++;
}

// Выводим кол-вом вхождений слов
System.out.println(biMap); // напечатает {ball=мяч, snow=снег, one=один, two=два, three=три}- в произвольном
// порядке
// Выводим все уникальные слова
System.out.println(biMap.keySet()); // напечатает [ball, snow, one, two, three]- в произвольном порядке
System.out.println(biMap.values()); // напечатает [мяч, снег, один, два, три]- в произвольном порядке

// Выводим перевод по каждому слову
```

```

System.out.println("one = " + biMap.get("one")); // напечатает one = один
System.out.println("two = " + biMap.get("two")); // напечатает two = два
System.out.println("мяч = " + biMap.getKey("мяч")); // напечатает мяч = ball
System.out.println("снег = " + biMap.getKey("снег")); // напечатает снег = snow
System.out.println("empty = " + biMap.get("empty")); // напечатает empty = null

// Выводим общее количество переводов в словаре
System.out.println(biMap.size()); //напечатает 5

```

3. С помощью HashBiMap от GS Collections:

▼ Используем HashBiMap от GS Collections

```

String[] englishWords = {"one", "two", "three", "ball", "snow"};
String[] russianWords = {"один", "два", "три", "мяч", "снег"};

// Создаем Multiset
MutableBiMap<String, String> biMap = new HashBiMap(englishWords.length);
// создаем англо-русский словарь
int i = 0;
for(String englishWord: englishWords) {
    biMap.put(englishWord, russianWords[i]);
    i++;
}

// Выводим кол-вом вхождений слов
System.out.println(biMap); // напечатает {two=два, ball=мяч, one=один, snow=снег, three=три} - в
произвольном порядке
// Выводим все уникальные слова
System.out.println(biMap.keySet()); // напечатает [snow, two, one, three, ball] - в произвольном порядке
System.out.println(biMap.values()); // напечатает [два, мяч, один, снег, три] - в произвольном порядке

// Выводим перевод по каждому слову
System.out.println("one = " + biMap.get("one")); // напечатает one = один
System.out.println("two = " + biMap.get("two")); // напечатает two = два
System.out.println("мяч = " + biMap.inverse().get("мяч")); // напечатает мяч = ball
System.out.println("снег = " + biMap.inverse().get("снег")); // напечатает снег = snow
System.out.println("empty = " + biMap.get("empty")); // напечатает empty = null

// Выводим общее количество переводов в словаре
System.out.println(biMap.size()); //напечатает 5

```

4. Ну и наконец, посмотрим как можно сделать тоже самое в чистом JDK

▼ Используем две HashMap для эмуляции BiMap

```

String[] englishWords = {"one", "two", "three", "ball", "snow"};
String[] russianWords = {"один", "два", "три", "мяч", "снег"};

// Создаем аналог BiMap
Map<String, String> biMapKeys = new HashMap(englishWords.length);
Map<String, String> biMapValues = new HashMap(russianWords.length);
// создаем англо-русский словарь
int i = 0;
for(String englishWord: englishWords) {
    biMapKeys.put(englishWord, russianWords[i]);
    biMapValues.put(russianWords[i], englishWord);
    i++;
}

// Выводим кол-вом вхождений слов

```



```
System.out.println(biMapKeys); // напечатает {ball=мяч, two=два, three=три, snow=снег, one=один}- в произвольном порядке
// Выводим все уникальные слова

System.out.println(biMapKeys.keySet()); // напечатает [ball, two, three, snow, one] - в произвольном порядке
System.out.println(biMapValues.keySet()); // напечатает [два, три, мяч, снег, один] - в произвольном порядке

// Выводим перевод по каждому слову

System.out.println("one = " + biMapKeys.get("one")); // напечатает one = один
System.out.println("two = " + biMapKeys.get("two")); // напечатает two = два
System.out.println("мяч = " + biMapValues.get("мяч")); // напечатает мяч = ball
System.out.println("снег = " + biMapValues.get("снег")); // напечатает снег = snow
System.out.println("empty = " + biMapValues.get("empty")); // напечатает empty = null

// Выводим общее количество переводов в словаре

System.out.println(biMapKeys.size()); //напечатает 5
```

V. Сравнение операций работы с коллекциями

Давайте кратко посмотрим какие дополнительные методы, операции и алгоритмы предлагают альтернативные библиотеки по сравнению со стандартными возможностями JDK. Цель этого, естественно, не перечислить все возможные методы всех библиотек (это невозможно), а скорее дать краткое представление о философии и синтаксисе разных библиотек, чтобы каждый мог выбрать то что больше нравится именно ему.

5.1 Сравним создание коллекций с помощью методов различных библиотек.

Guava и gs-collections предлагают создание коллекций через статические методы утилиты вместо использование new, давайте посмотрим насколько это удобнее обычного способа jdk.

5.1.1) Создание списка (List)

Название	JDK	guava	gs-collections
Создание пустого списка	new ArrayList<>()	Lists.newArrayList()	FastList.newList()
Создание списка из значений	Arrays.asList(«1», «2», «3»)	Lists.newArrayList(«1», «2», «3»)	FastList.newListWith(«1», «2», «3»)
Создать список с определенным capacity	new ArrayList<>(100)	Lists.newArrayListWithCapacity(100)	FastList.newList(100)
Создать список из любой коллекции	new ArrayList<>(collection)	Lists.newArrayList(collection)	FastList.newList(collection)
Создать список из любого Iterable	-	Lists.newArrayList(iterable)	FastList.newList(iterable)
Создать список из Iterator'a	-	Lists.newArrayList(iterator)	-
Создать список из массива	Arrays.asList(array)	Lists.newArrayList(array)	FastList.newListWith(array)
Создать список с помощью фабрики	—	—	FastList.newWithNValues(10, () -> «1»)

▼ Примеры создания списка

```
// Простое создание пустых коллекций

List<String> emptyGuava = Lists.newArrayList(); // с помощью guava
List<String> emptyJDK = new ArrayList<>(); // аналог JDK
MutableList<String> emptyGS = FastList.newList(); // с помощью gs

// Создать список ровно со 100 элементами

List < String > exactly100 = Lists.newArrayListWithCapacity(100); // с помощью guava
List<String> exactly100JDK = new ArrayList<>(100); // аналог JDK
MutableList<String> empty100GS = FastList.newList(100); // с помощью gs

// Создать список в котором ожидается около 100 элементов (чуть больше чем 100)
```

```

List<String> approx100 = Lists.newArrayListWithExpectedSize(100); // с помощью guava
List<String> approx100JDK = new ArrayList<>(115); // аналог JDK
MutableList<String> approx100GS = FastList.newList(115); // с помощью gs

// Создать список из заданных элементов
List<String> withElements = Lists.newArrayList("alpha", "beta", "gamma"); // с помощью guava
List<String> withElementsJDK = Arrays.asList("alpha", "beta", "gamma"); // аналог JDK
MutableList<String> withElementsGS = FastList.newListWith("alpha", "beta", "gamma"); // с помощью gs

System.out.println(withElements);
System.out.println(withElementsJDK);
System.out.println(withElementsGS);

// Создать список из любого объекта Iterable интерфейса (то есть любой коллекции)
Collection<String> collection = new HashSet<>(3);
collection.add("1");
collection.add("2");
collection.add("3");

List<String> fromIterable = Lists.newArrayList(collection); // с помощью guava
List<String> fromIterableJDK = new ArrayList<>(collection); // аналог JDK
MutableList<String> fromIterableGS = FastList.newList(collection); // с помощью gs

System.out.println(fromIterable);
System.out.println(fromIterableJDK);
System.out.println(fromIterableGS);
/* обратите внимание у JDK необходим объект Collection интерфейса, у guava и gs достаточно Iterable */

// Создать список из Iterator'a
Iterator<String> iterator = collection.iterator();
List<String> fromIterator = Lists.newArrayList(iterator); // с помощью guava, аналога в JDK нет
System.out.println(fromIterator);

// Создать список из массива
String[] array = {"4", "5", "6"};
List<String> fromArray = Lists.newArrayList(array); // с помощью guava
List<String> fromArrayJDK = Arrays.asList(array); // аналог JDK
MutableList<String> fromArrayGS = FastList.newListWith(array); // с помощью gs

System.out.println(fromArray);
System.out.println(fromArrayJDK);
System.out.println(fromArrayGS);

// Создать список из с помощью фабрики
MutableList<String> fromFabricGS = FastList.newWithNValues(10, () -> String.valueOf(Math.random())); // с помощью gs
System.out.println(fromFabricGS);

```

5.1.2) Создание множества (set)

Название	JDK	guava	gs-collections
Создание пустого множества	<code>new HashSet<>()</code>	<code>Sets.newHashSet()</code>	<code>UnifiedSet.newSet()</code>
Создать множество из заданных элементов	<code>new HashSet<>(Arrays.asList(«alpha», «beta», «gamma»))</code>	<code>Sets.newHashSet(«alpha», «beta», «gamma»)</code>	<code>UnifiedSet.newSetWith(«alpha», «beta», «gamma»)</code>
Создать множество из любой коллекции	<code>new HashSet<>(collection)</code>	<code>Sets.newHashSet(collection)</code>	<code>UnifiedSet.newSet(collection)</code>
Создать множество из любого Iterable	-	<code>Sets.newHashSet(iterable)</code>	<code>UnifiedSet.newSet(iterable)</code>
Создать множество из Iterator'a	-	<code>Sets.newHashSet(iterator);</code>	-
Создать множество из массива	<code>new HashSet<>(Arrays.asList(array))</code>	<code>Sets.newHashSet(array)</code>	<code>UnifiedSet.newSetWith(array)</code>

▼ [Примеры создания множества](#)

```
// Простое создание пустых коллекций
Set<String> emptyGuava = Sets.newHashSet(); // с помощью guava
Set<String> emptyJDK = new HashSet<>(); // аналог JDK
Set<String> emptyGS = UnifiedSet.newSet(); // с помощью gs

// Создать множество в котором ожидается около 100 элементов (чуть больше чем 100)
Set<String> approx100 = Sets.newHashSetWithExpectedSize(100); // с помощью guava
Set<String> approx100JDK = new HashSet<>(130); // аналог JDK
Set<String> approx100GS = UnifiedSet.newSet(130); // с помощью gs

// Создать множество из заданных элементов
Set<String> withElements = Sets.newHashSet("alpha", "beta", "gamma"); // с помощью guava
Set<String> withElementsJDK = new HashSet<>(Arrays.asList("alpha", "beta", "gamma")); // аналог JDK
Set<String> withElementsGS = UnifiedSet.newSetWith("alpha", "beta", "gamma"); // с помощью gs

System.out.println(withElements);
System.out.println(withElementsJDK);
System.out.println(withElementsGS);

// Создать множество из любого объекта Iterable интерфейса (то есть любой коллекции)
Collection<String> collection = new ArrayList<>(3);
collection.add("1");
collection.add("2");
collection.add("3");

Set<String> fromIterable = Sets.newHashSet(collection); // с помощью guava
Set<String> fromIterableJDK = new HashSet<>(collection); // аналог JDK
Set<String> fromIterableGS = UnifiedSet.newSet(collection); // с помощью gs

System.out.println(fromIterable);
System.out.println(fromIterableJDK);
System.out.println(fromIterableGS);
/* обратите внимание у JDK необходим объект Collection интерфейса, у guava достаточно Iterable */

// Создать множество из Iterator'a
Iterator<String> iterator = collection.iterator();
Set<String> fromIterator = Sets.newHashSet(iterator); // с помощью guava, аналога в JDK нет
System.out.println(fromIterator);

// Создать множество из массива
String[] array = {"4", "5", "6"};
Set<String> fromArray = Sets.newHashSet(array); // с помощью guava
Set<String> fromArrayJDK = new HashSet<>(Arrays.asList(array)); // аналог JDK
Set<String> fromArrayGS = UnifiedSet.newSetWith(array); // с помощью gs

System.out.println(fromArray);
System.out.println(fromArrayJDK);
System.out.println(fromArrayGS);
```

5.1.3) Создание Map

Название	JDK	guava	gs-collections
Создание пустой map'ы	new HashMap<>()	Maps.newHashMap()	UnifiedMap.newMap()
Создать map'у с определенным capacity	new HashMap<>(130)	Maps.newHashMapWithExpectedSize(100)	UnifiedMap.newMap(130)
Создать map'у из другой map'ы	new HashMap<>(map)	Maps.newHashMap(map)	UnifiedMap.newMap(map)
Создать map'у из ключей	-	-	UnifiedMap.newWithKeyValues(«1», «a», «2», «b»)

▼ [Примеры создания map](#)

```
// Простое создание пустых коллекций
Map<String, String> emptyGuava = Maps.newHashMap(); // с помощью guava
Map<String, String> emptyJDK = new HashMap<>(); // аналог JDK
Map<String, String> emptyGS = UnifiedMap.newMap(); // с помощью gs

// Создать map'у в котором ожидается около 100 элементов (чуть больше чем 100)
Map<String, String> approx100 = Maps.newHashMapWithExpectedSize(100); // с помощью guava
Map<String, String> approx100JDK = new HashMap<>(130); // аналог JDK
Map<String, String> approx100GS = UnifiedMap.newMap(130); // с помощью gs

// Создать map'у из другой map'ы
Map<String, String> map = new HashMap<>(3);
map.put(«k1», «v1»);
map.put(«k2», «v2»);
Map<String, String> withMap = Maps.newHashMap(map); // с помощью guava
Map<String, String> withMapJDK = new HashMap<>(map); // аналог JDK
Map<String, String> withMapGS = UnifiedMap.newMap(map); // с помощью gs

System.out.println(withMap);
System.out.println(withMapJDK);
System.out.println(withMapGS);

// Создать map'у из ключей
Map<String, String> withKeys = UnifiedMap.newWithKeysValues(«1», «a», «2», «b»);
System.out.println(withKeys);
```

5.2 Сравним методы поиска из различных библиотек.

Название	JDK	guava	apache	gs-collections
Найти количество вхождений объекта	Collections.frequency(collection, «1»)	Iterables.frequency(iterable, «1»)	CollectionUtils.cardinality(«1», iterable)	mutableCollection.count(each) -> «a1».equals(each))
Вернуть первый элемент коллекции или значение по умолчанию	collection.stream().findFirst().orElse(«1»)	Iterables.getFirst(iterable, «1»)	CollectionUtils.get(iterable, 0)	orderedIterable.getFirst()
Вернуть последний элемент коллекции или значение по умолчанию	collection.stream().skip(collection.size()-1).findFirst().orElse(«1»);	Iterables.getLast(iterable, «1»)	CollectionUtils.get(collection, collection.size()-1)	orderedIterable.getLast()
Вернуть максимальный элемент	Collections.max(collection)	Ordering.natural().max(iterable)	-	orderedIterable.max()
Вернуть минимальный элемент	Collections.min(collection)	Ordering.natural().min(iterable)	-	orderedIterable.min()
Вернуть единственный элемент коллекции		Iterables.getOnlyElement(iterable)	CollectionUtils.extractSingleton(collection)	
Найти элемент в отсортированном списке	Collections.binarySearch(list, «13»)	Ordering.natural().binarySearch(list, «13»)		mutableList.binarySearch(«13»)
Найти элемент в неотсортированной коллекции	collection.stream().filter(«13»::equals).findFirst().get()	Iterables.find(iterable, «13»::equals)	CollectionUtils.find(iterable, «13»::equals)	mutableList.select(«13»::equals).get(0)
Выбрать все элементы по условию	collection.stream().filter((s) -> s.contains(«1»)).collect(Collectors.toList())	Iterables.filter(iterable, (s) -> s.contains(«1»))	CollectionUtils.select(iterable, (s) -> s.contains(«1»))	mutableCollection.select((s) -> s.contains(«1»))

Обратите внимание что методы разных библиотек работают с разными сущностями, это можно определить по названию переменных: collection — любая реализация интерфейса Collection, iterable — интерфейса Iterable, list — интерфейса List, orderedIterable и mutableList соответствующих интерфейсов в GS (orderedIterable — интерфейс для всех коллекций у которых определен порядок элементов, mutableList — интерфейс для любых изменяемых списков)

Примеры:

▼ [1\) Найти количество вхождений объекта](#)

```

Collection<String> collection = Lists.newArrayList("a1", "a2", "a3", "a1");
Iterable<String> iterable = collection;
MutableCollection<String> collectionGS = FastList.newListWith("a1", "a2", "a3", "a1");

// Вернуть количество вхождений объекта
int i1 = Iterables.frequency(iterable, "a1"); // с помощью guava
int i2 = Collections.frequency(collection, "a1"); // с помощью JDK
int i3 = CollectionUtils.cardinality("a1", iterable); // с помощью Apache
int i4 = collectionGS.count((s) -> "a1".equals(s));
long i5 = collection.stream().filter((s) -> "a1".equals(s)).count(); // с помощью stream JDK

System.out.println("count = " + i1 + ":" + i2 + ":" + i3 + ":" + i4 + ":" + i5); // напечатает count = 2:2:2:2:2

```

▼ 2) Вернуть первый элемент коллекции

```

Collection<String> collection = Lists.newArrayList("a1", "a2", "a3", "a1");
OrderedIterable<String> orderedIterable = FastList.newListWith("a1", "a2", "a3", "a1");
Iterable<String> iterable = collection;

// вернуть первый элемент коллекции
Iterator<String> iterator = collection.iterator(); // с помощью JDK
String jdk = iterator.hasNext() ? iterator.next(): "1";
String guava = Iterables.getFirst(iterable, "1"); // с помощью guava
String apache = CollectionUtils.get(iterable, 0); // с помощью Apache
String gs = orderedIterable.getFirst(); // с помощью GS
String stream = collection.stream().findFirst().orElse("1"); // с помощью Stream API
System.out.println("first = " + jdk + ":" + guava + ":" + apache + ":" + gs + ":" + stream); // напечатает first = a1:a1:a
1:a1:a1

```

▼ 3) Вернуть последний элемент коллекции

```

Collection<String> collection = Lists.newArrayList("a1", "a2", "a3", "a8");
OrderedIterable<String> orderedIterable = FastList.newListWith("a1", "a2", "a3", "a8");
Iterable<String> iterable = collection;

// вернуть последний элемент коллекции
Iterator<String> iterator = collection.iterator(); // с помощью JDK
String jdk = "1";
while(iterator.hasNext()) {
    jdk = iterator.next();
}
String guava = Iterables.getLast(iterable, "1"); // с помощью guava
String apache = CollectionUtils.get(collection, collection.size()-1); // с помощью Apache
String gs = orderedIterable.getLast(); // с помощью GS
String stream = collection.stream().skip(collection.size()-1).findFirst().orElse("1"); // с помощью Stream API
System.out.println("last = " + jdk + ":" + guava + ":" + apache + ":" + gs + ":" + stream); // напечатает last = a8:a8:a8:
a8:a8

```

▼ 4) Вернуть максимальный элемент

```

Collection<String> collection = Lists.newArrayList("5", "1", "3", "8", "4");
OrderedIterable<String> orderedIterable = FastList.newListWith("5", "1", "3", "8", "4");
Iterable<String> iterable = collection;

// вернуть максимальный элемент коллекции
String jdk = Collections.max(collection); // с помощью JDK
String gs = orderedIterable.max(); // с помощью GS
String guava = Ordering.natural().max(iterable); // с помощью guava

System.out.println("max = " + jdk + ":" + guava + ":" + gs); // напечатает max = 8:8:8

```

▼ 5) Вернуть минимальный элемент

```
Collection<String> collection = Lists.newArrayList("5", "1", "3", "8", "4");
OrderedIterable<String> orderedIterable = FastList.newListWith("5", "1", "3", "8", "4");
Iterable<String> iterable = collection;

// вернуть минимальный элемент коллекции
String jdk = Collections.min(collection); // с помощью JDK
String gs = orderedIterable.min(); // с помощью GS
String guava = Ordering.natural().min(iterable); // с помощью guava
System.out.println("min = " + jdk + ":" + guava + ":" + gs); // напечатает min = 1:1:1
```

▼ 6) вернуть единственный элемент коллекции

```
Collection<String> collection = Lists.newArrayList("a3");
OrderedIterable<String> orderedIterable = FastList.newListWith("a3");
Iterable<String> iterable = collection;

// вернуть единственный элемент коллекции
String guava = Iterables.getOnlyElement(iterable); // с помощью guava
String jdk = collection.iterator().next(); // с помощью JDK
String apache = CollectionUtils.extractSingleton(collection); // с помощью Apache
assert(orderedIterable.size() > 1); // с помощью GS
String gs = orderedIterable.getFirst();

System.out.println("single = " + jdk + ":" + guava + ":" + apache + ":" + gs); // напечатает single = a3:a3:a3:a3
```

▼ 7) найти элемент в отсортированном списке

```
List<String> list = Lists.newArrayList("2", "4", "13", "31", "43");
MutableList<String> mutableList = FastList.newListWith("2", "4", "13", "31", "43");

// найти элемент в отсортированном списке
int jdk = Collections.binarySearch(list, "13");
int guava = Ordering.natural().binarySearch(list, "13");
int gs = mutableList.binarySearch("13");

System.out.println("find = " + jdk + ":" + guava + ":" + gs); // напечатает find = 2:2:2
```

▼ 8) найти элемент в неотсортированной коллекции

```
Collection<String> collection = Lists.newArrayList("a1", "a2", "a3", "a1");
MutableCollection<String> orderedIterable = FastList.newListWith("a1", "a2", "a3", "a1");
Iterable<String> iterable = collection;

// вернуть третий элемент коллекции по порядку
String jdk = collection.stream().skip(2).findFirst().get(); // с помощью JDK
String guava = Iterables.get(iterable, 2); // с помощью guava
String apache = CollectionUtils.get(iterable, 2); // с помощью Apache
System.out.println("third = " + jdk + ":" + guava + ":" + apache); // напечатает third = a3:a3:a3
```

▼ 9) выбрать все элементы по условию

```
Collection<String> collection = Lists.newArrayList("2", "14", "3", "13", "43");
MutableCollection<String> mutableCollection = FastList.newListWith("2", "14", "3", "13", "43");
Iterable<String> iterable = collection;

// выбрать все элементы по шаблону
List<String> jdk = collection.stream().filter((s) -> s.contains("1")).collect(Collectors.toList()); // с помощью JDK
Iterable<String> guava = Iterables.filter(iterable, (s) -> s.contains("1")); // с помощью guava
```

```
Collection<String> apache = CollectionUtils.select(iterable, (s) -> s.contains("1")); // с помощью Apache
MutableCollection<String> gs = mutableCollection.select((s) -> s.contains("1")); // с помощью GS

System.out.println("select = " + jdk + ":" + guava + ":" + apache + ":" + gs); // напечатает select = [14, 13]:[14, 13]:[14, 13]:[14, 13]
```

5.3 Сравним методы сравнений, объединений и пересечений коллекций

Название	JDK	guava	apache	gs-collections
Проверить полное соответствие двух коллекций	collection1.containsAll(collection2)	Iterables.elementsEqual(iterable1, iterable2)	CollectionUtils.containsAll(collection1, collection2)	mutableCollection1.containsAll(mutableCollection2)
Наличие хотя бы одного общего элемента	!Collections.disjoint(collection1, collection2)	!Sets.intersection(set1, set2).isEmpty()	CollectionUtils.containsAny(collection1, collection2)	!mutableSet1.intersect(mutableSet2).isEmpty()
Найти все общие элементы (пересечение)	Set<T> result = new HashSet<>(set1); result.retainAll(set2)	Sets.intersection(set1, set2)	CollectionUtils.intersection(collection1, collection2)	mutableSet1.intersect(mutableSet2)
Отсутствие общих элементов	Collections.disjoint(collection1, collection2)	Sets.intersection(set1, set2).isEmpty()	!CollectionUtils.containsAny(collection1, collection2)	mutableSet1.intersect(mutableSet2).isEmpty()
Найти все элементы, которые есть в одной коллекции и нет в другой (difference)	Set<T> result = new HashSet<>(set1); result.removeAll(set2)	Sets.difference(set1, set2)	CollectionUtils.removeAll(collection1, collection2)	mutableSet1.difference(mutableSet2)
Найти все различные элементы (symmetric difference)		Sets.symmetricDifference(set1, set2)	CollectionUtils.disjunction(collection1, collection2)	mutableSet1.symmetricDifference(mutableSet2)
Получить объединение двух коллекций	Set<T> result = new HashSet<>(set1); result.addAll(set2)	Sets.union(set1, set2)	CollectionUtils.union(collection1, collection2)	mutableSet1.union(mutableSet2)

Примеры:

▼ 1) Проверить полное соответствие двух коллекций

```
Collection<String> collection1 = Lists.newArrayList("a1", "a2", "a3", "a1");
Collection<String> collection2 = Lists.newArrayList("a1", "a2", "a3", "a1");
Iterable<String> iterable1 = collection1;
Iterable<String> iterable2 = collection2;
MutableCollection<String> mutableCollection1 = FastList.newListWith("a1", "a2", "a3", "a1");
MutableCollection<String> mutableCollection2 = FastList.newListWith("a1", "a2", "a3", "a1");

// Проверить полное соответствие двух коллекций
boolean jdk = collection1.containsAll(collection2); // с помощью JDK
boolean guava = Iterables.elementsEqual(iterable1, iterable2); // с помощью guava
boolean apache = CollectionUtils.containsAll(collection1, collection2); // с помощью Apache
boolean gs = mutableCollection1.containsAll(mutableCollection2); // с помощью GS

System.out.println("containsAll = " + jdk + ":" + guava + ":" + apache + ":" + gs); // напечатает containsAll = true:true:true:true
```

▼ 2) Проверить наличие хотя бы одного общего элемента у двух коллекций

```
Collection<String> collection1 = Lists.newArrayList("a1", "a2", "a3", "a1");
Collection<String> collection2 = Lists.newArrayList("a4", "a8", "a3", "a5");
Set<String> set1 = Sets.newHashSet("a1", "a2", "a3", "a1");
Set<String> set2 = Sets.newHashSet("a4", "a8", "a3", "a5");
MutableSet<String> mutableSet1 = UnifiedSet.newSetWith("a1", "a2", "a3", "a1");
MutableSet<String> mutableSet2 = UnifiedSet.newSetWith("a4", "a8", "a3", "a5");

// Проверить наличие хотя бы одного общего элемента у двух коллекций
boolean jdk = !Collections.disjoint(collection1, collection2); // с помощью JDK
```

```

boolean guava = !Sets.intersection(set1, set2).isEmpty(); // с помощью guava
boolean apache = CollectionUtils.containsAny(collection1, collection2); // с помощью Apache
boolean gs = !mutableSet1.intersect(mutableSet2).isEmpty(); // с помощью GS
System.out.println("containsAny = " + jdk + ":" + guava + ":" + apache + ":" + gs); // напечатает containsAny = true:true:
true:true

```

▼ 3) Найти все общие элементы (пересечение) у двух коллекций

```

Collection<String> collection1 = Lists.newArrayList("a1", "a2", "a3", "a1");
Collection<String> collection2 = Lists.newArrayList("a4", "a8", "a3", "a5");
Set<String> set1 = Sets.newHashSet("a1", "a2", "a3", "a1");
Set<String> set2 = Sets.newHashSet("a4", "a8", "a3", "a5");
MutableSet<String> mutableSet1 = UnifiedSet.newSetWith("a1", "a2", "a3", "a1");
MutableSet<String> mutableSet2 = UnifiedSet.newSetWith("a4", "a8", "a3", "a5");

// Найти все общие элементы у двух коллекций
Set<String> jdk = new HashSet<>(set1); // с помощью JDK
jdk.retainAll(set2);
Set<String> guava = Sets.intersection(set1, set2); // с помощью guava
Collection<String> apache = CollectionUtils.intersection(collection1, collection2); // с помощью Apache
Set<String> gs = mutableSet1.intersect(mutableSet2); // с помощью GS
System.out.println("intersect = " + jdk + ":" + guava + ":" + apache + ":" + gs); // напечатает intersect = [a3]:[a3]:[a
3]:[a3]

```

▼ 4) Найти все элементы, которые есть в одной коллекции и нет в другой (difference)

```

Collection<String> collection1 = Lists.newArrayList("a2", "a3");
Collection<String> collection2 = Lists.newArrayList("a8", "a3", "a5");
Set<String> set1 = Sets.newHashSet("a2", "a3");
Set<String> set2 = Sets.newHashSet("a8", "a3", "a5");
MutableSet<String> mutableSet1 = UnifiedSet.newSetWith("a2", "a3");
MutableSet<String> mutableSet2 = UnifiedSet.newSetWith("a8", "a3", "a5");

// Найти все элементы, которые есть в одной коллекции и нет в другой (difference)
Set<String> jdk = new HashSet<>(set1); // с помощью JDK
jdk.removeAll(set2);
Set<String> guava = Sets.difference(set1, set2); // с помощью guava
Collection<String> apache = CollectionUtils.removeAll(collection1, collection2); // с помощью Apache
Set<String> gs = mutableSet1.difference(mutableSet2); // с помощью GS
System.out.println("difference = " + jdk + ":" + guava + ":" + apache + ":" + gs); // напечатает difference = [a2]:[a2]:[a
2]:[a2]

```

▼ 5) Найти все различные элементы (symmetric difference) у двух коллекций

```

Collection<String> collection1 = Lists.newArrayList("a2", "a3");
Collection<String> collection2 = Lists.newArrayList("a8", "a3", "a5");
Set<String> set1 = Sets.newHashSet("a2", "a3");
Set<String> set2 = Sets.newHashSet("a8", "a3", "a5");
MutableSet<String> mutableSet1 = UnifiedSet.newSetWith("a2", "a3");
MutableSet<String> mutableSet2 = UnifiedSet.newSetWith("a8", "a3", "a5");

// Найти все различные элементы (symmetric difference) у двух коллекций
Set<String> intersect = new HashSet<>(set1); // с помощью JDK
intersect.retainAll(set2);

Set<String> jdk = new HashSet<>(set1);
jdk.addAll(set2);
jdk.removeAll(intersect);

Set<String> guava = Sets.symmetricDifference(set1, set2); // с помощью guava
Collection<String> apache = CollectionUtils.disjunction(collection1, collection2); // с помощью Apache
Set<String> gs = mutableSet1.symmetricDifference(mutableSet2); // с помощью GS

```



```
System.out.println("symmetricDifference = " + jdk + ":" + guava + ":" + apache + ":" + gs); // напечатает symmetricDifference = [a2, a5, a8]:[a2, a5, a8]:[a2, a5, a8]:[a2, a5, a8]
```

▼ 6) Получить объединение двух коллекций

```
Set<String> set1 = Sets.newHashSet("a1", "a2");
Set<String> set2 = Sets.newHashSet("a4");
MutableSet<String> mutableSet1 = UnifiedSet.newSetWith("a1", "a2");
MutableSet<String> mutableSet2 = UnifiedSet.newSetWith("a4");
Collection<String> collection1 = set1;
Collection<String> collection2 = set2;
// Получить объединение двух коллекций
Set<String> jdk = new HashSet<>(set1); // с помощью JDK
jdk.addAll(set2);
Set<String> guava = Sets.union(set1, set2); // с помощью guava
Collection<String> apache = CollectionUtils.union(collection1, collection2); // с помощью Apache
Set<String> gs = mutableSet1.union(mutableSet2); // с помощью GS
System.out.println("union = " + jdk + ":" + guava + ":" + apache + ":" + gs); // напечатает union = [a1, a2, a4]:[a1, a2, a4]:[a1, a2, a4]:[a1, a2, a4]
```

5.4 Сравним методы изменения коллекции

Название	JDK	guava	apache	gs-collections
Сортировка коллекции	<code>Collections.sort(list);</code>	<code>Ordering.natural().sortedCopy(iterable)</code>		<code>mutableList.sortThis()</code>
Удалить все элементы соответствующие условию	<code>collection.removeIf((s) -> s.contains(«1»))</code>	<code>Iterables.removeIf(iterable, (s) -> s.contains(«1»))</code>	<code>CollectionUtils.filter(iterable, (s) -> !s.contains(«1»))</code>	<code>mutableCollection.removeIf((Predicate<String>) (s) -> s.contains(«1»))</code>
Удалить все элементы не соответствующие условию	<code>collection.removeIf((s) -> !s.contains(«1»))</code>	<code>Iterables.removeIf(iterable, (s) -> !s.contains(«1»))</code>	<code>CollectionUtils.filter(iterable, (s) -> s.contains(«1»))</code>	<code>mutableCollection.removeIf((Predicate<String>) (s) -> !s.contains(«1»))</code>
Изменить все элементы коллекции	<code>collection.stream().map((s) -> s + "_1").collect(Collectors.toList())</code>	<code>Iterables.transform(iterable, (s) -> s + "_1")</code>	<code>CollectionUtils.transform(collection, (s) -> s + "_1")</code>	<code>mutableCollection.collect((s) -> s + "_1")</code>
Изменить свойства каждого элемента	<code>collection.stream().forEach((s) -> s.append("_1"))</code>	<code>Iterables.transform(iterable, (s) -> s.append("_1"))</code>	<code>CollectionUtils.transform(collection, (s) -> s.append("_1"))</code>	<code>mutableCollection.forEach((Procedure<StringBuilder>) (s) -> s.append("_1"))</code>

▼ 1) Сортировка коллекции

```
List<String> jdk = Lists.newArrayList("a1", "a2", "a3", "a1");
Iterable<String> iterable = jdk;
MutableList<String> gs = FastList.newList(jdk);

// Сортировка коллекции
Collections.sort(jdk); // с помощью jdk
List<String> guava = Ordering.natural().sortedCopy(iterable); // с помощью guava
gs.sortThis(); // с помощью gs

System.out.println("sort = " + jdk + ":" + guava + ":" + gs); // напечатает sort = [a1, a1, a2, a3]:[a1, a1, a2, a3]:[a1, a1, a2, a3]
```

▼ 2) Удалить все элементы соответствующие условию

```
Collection<String> jdk = Lists.newArrayList("a1", "a2", "a3", "a1");
Iterable<String> guava = Lists.newArrayList(jdk);
Iterable<String> apache = Lists.newArrayList(jdk);
```

```

MutableCollection<String> gs = FastList.newList(jdk);

// Удалить все элементы соответствующие условию
jdk.removeIf((s) -> s.contains("1")); // с помощью jdk
Iterables.removeIf(guava, (s) -> s.contains("1")); // с помощью guava
CollectionUtils.filter(apache, (s) -> !s.contains("1")); // с помощью apache
gs.removeIf((Predicate<String>) (s) -> s.contains("1")); // с помощью gs

System.out.println("removeIf = " + jdk + ":" + guava + ":" + apache + ":" + gs); // напечатает removeIf = [a2, a3]:[a2, a3]:[a2, a3]:[a2, a3]

```

▼ 3) Удалить все элементы не соответствующие условию

```

Collection<String> jdk = Lists.newArrayList("a1", "a2", "a3", "a1");
Iterable<String> guava = Lists.newArrayList(jdk);
Iterable<String> apache = Lists.newArrayList(jdk);
MutableCollection<String> gs = FastList.newList(jdk);

// Удалить все элементы не соответствующие условию
jdk.removeIf((s) -> !s.contains("1")); // с помощью jdk
Iterables.removeIf(guava, (s) -> !s.contains("1")); // с помощью guava
CollectionUtils.filter(apache, (s) -> s.contains("1")); // с помощью apache
gs.removeIf((Predicate<String>) (s) -> !s.contains("1")); // с помощью gs

System.out.println("retainIf = " + jdk + ":" + guava + ":" + apache + ":" + gs); // напечатает retainIf = [a1, a1]:[a1, a1]:[a1, a1]:[a1, a1]

```

▼ 4) Преобразовать все элементы коллекции

```

Collection<String> collection = Lists.newArrayList("a1", "a2", "a3", "a1");
Iterable<String> iterable = collection;
Collection<String> apache = Lists.newArrayList(collection);
MutableCollection<String> mutableCollection = FastList.newList(collection);

// Преобразовать все элементы коллекции в другие элементы
List<String> jdk = collection.stream().map((s) -> s + "_1").collect(Collectors.toList()); // с помощью jdk
Iterable<String> guava = Iterables.transform(iterable, (s) -> s + "_1"); // с помощью guava
CollectionUtils.transform(apache, (s) -> s + "_1"); // с помощью apache
MutableCollection<String> gs = mutableCollection.collect((s) -> s + "_1"); // с помощью gs

System.out.println("transform = " + jdk + ":" + guava + ":" + apache + ":" + gs); // напечатает transform = [a1_1, a2_1, a3_1, a1_1]:[a1_1, a2_1, a3_1, a1_1]:[a1_1, a2_1, a3_1, a1_1]:[a1_1, a2_1, a3_1, a1_1]

```

▼ 5) Изменить свойства каждого элемента коллекции

```

Collection<StringBuilder> jdk = Lists.newArrayList(new StringBuilder("a1"), new StringBuilder("a2"), new StringBuilder("a3"));
Iterable<StringBuilder> iterable = Lists.newArrayList(new StringBuilder("a1"), new StringBuilder("a2"), new StringBuilder("a3"));
Collection<StringBuilder> apache = Lists.newArrayList(new StringBuilder("a1"), new StringBuilder("a2"), new StringBuilder("a3"));
MutableCollection<StringBuilder> gs = FastList.newListWith(new StringBuilder("a1"), new StringBuilder("a2"), new StringBuilder("a3"));

// Изменить свойства каждого элемента коллекции
jdk.stream().forEach((s) -> s.append("_1")); // с помощью jdk
Iterable<StringBuilder> guava = Iterables.transform(iterable, (s) -> s.append("_1")); // с помощью guava
CollectionUtils.transform(apache, (s) -> s.append("_1")); // с помощью apache
gs.forEach((Procedure<StringBuilder>) (s) -> s.append("_1")); // с помощью gs

System.out.println("change = " + jdk + ":" + guava + ":" + apache + ":" + gs); // changeAll = [a1_1, a2_1, a3_1]:[a1_1, a2_1, a3_1]:[a1_1, a2_1, a3_1]:[a1_1, a2_1, a3_1]

```

VI. Сравнение стандартных и альтернативных коллекций

5.1 Какие вообще бывают коллекции

Для начала рассмотрим какие основные коллекции встречаются в программировании (не только в Java, а вообще):

- 1) **Вектор (Список)** — элементы коллекции упорядочены, можно обойти все элементы по очереди или обратиться по индексу,
 - *Массив* — реализация вектора, когда данные находятся в памяти непосредственно друг за другом
 - *Динамический массив* — реализация массива, когда размер массива может увеличиваться во время выполнения,
 - *Односвязный список* — реализация списка, когда каждый элемент списка содержит значение и ссылку на следующий элемент, в отличие от массива более структурно гибко,
 - *Двусвязный список* — реализация списка когда каждый элемент списка содержит значение и ссылку на следующий и предыдущий элемент,
- 2) **Стек (Stack)** — коллекция, реализующая принцип хранения «LIFO» («последним пришёл — первым вышел»). В стеке постоянно доступен элемент добавленный последним, если он ещё не удален/извлечен.
- 3) **Очередь (Queue)** — коллекция, реализующая принцип хранения «FIFO» («первым пришёл — первым вышел»). В очереди постоянно доступен только добавлен самым первым из имеющихся и ещё не удален/извлечен.
- 4) **Двухсторонняя очередь (Double-ended queue)** — очередь, которая позволяет добавлять и извлекать данные и с начала очереди и с конца.
- 5) **Очередь с приоритетом (англ. priority queue)** — очередь, позволяющая добавить новый элемент и извлечь максимум. Все данные хранятся в порядке убывания приоритета,
 - *куча (heap)* — одна из реализаций очереди с приоритетом, с помощью дерева,
- 6) **Ассоциативный массив (словарь), (Associative array, Dictionary)** — неупорядоченная коллекция, хранящая пары «ключ — значение»
 - *Хеш-таблица (hashtable)* — реализация ассоциативного массива, построенная на вычислении хеша значения,
 - *Хеш-таблица со связями один ко многим (Multimap или multihash)* — реализация Хеш-таблицы, которая хранит отношение ключ и много значений,
 - *Двух-сторонняя хеш-таблица (bi-map)* — реализация Хеш-таблицы, которая позволяет получать как значение по ключу, так и ключ по значению,
 - *Упорядоченная хеш-таблица (hashtable)* — хеш-таблица, возвращающая элементы в порядке добавления,
 - *Отсортированная хеш-таблица (hashtable)* — хеш-таблица, возвращающая элементы отсортированным порядке,
- 7) **Множество** — неупорядоченная коллекция, хранящая набор уникальных значений и поддерживающая аналогичные операциям с математическими множествами,
 - *Мультимножество* — неупорядоченная коллекция, аналогичная множеству, но допускающая наличие в коллекции одновременно двух и более одинаковых значений,
 - *Упорядоченное множество* — коллекция, аналогичная множеству, но возвращает элементы в порядке добавления,
 - *Отсортированное множество* — коллекция, аналогичная множеству, но возвращает элементы в отсортированном порядке,
- 8) **Битовый массив** — то есть массив значений 1 или 0,
- 9) **Множество закрытых или открытых отрезков** — то есть структура хранящая и работающая с геометрическими интервалами,
- 10) **Деревья** — структура данных, хранящая данные в виде дерева,
- 11) **Кеши** — коллекции для работы с устаревающими за определенное время данными,

Давайте посмотрим какие из данных сущностей соответствуют каким коллекциям и интерфейсам Java и альтернативных библиотек:

Внимание: если таблица не помещается целиком, попробуйте уменьшить масштаб страницы или открыть в другом браузере.

Название	List	Set	Map	Query Dequery и т.п. в JDK	guava	apache	gs-collections
1) Вектор (Список) <i>Динамический массив</i>	ArrayList	LinkedHashSet	LinkedHashMap	ArrayDeque		TreeList	FastList
<i>Двусвязный список</i>	LinkedList	LinkedHashSet	LinkedHashMap	LinkedList		NodeCachingLinkedList	
2) Стек (Stack)	LinkedList			ArrayDeque			ArrayStack
3) Очередь (Queue)	LinkedList			ArrayDeque		CircularFifoQueue	
4) Двухсторонняя очередь	LinkedList			ArrayDeque			
5) Очередь с приоритетом				PriorityQueue			

куча (heap)							
6) Ассоциативный массив (словарь)			HashMap			HashMap	UnifiedMap
Хеш-таблица (hashtable)		HashSet	HashMap			HashMap	UnifiedMap
Хеш-таблица со связями один ко многим					Multimap	MultiMap	Multimap
Двух-сторонняя хеш-таблица					HashBiMap	BidiMap	HashBiMap
Упорядоченная хеш-таблица		LinkedHashSet	LinkedHashMap			LinkedMap	
Отсортированная хеш-таблица		TreeSet	TreeMap			PatriciaTrie	TreeSortedMap
7) Множество	HashSet						UnifiedSet
Мультимножество					HashMultiset	HashBag	HashBag
Упорядоченное множество		LinkedHashSet					
Отсортированное множество		TreeSet				PatriciaTrie	TreeSortedSet
8) Битовый массив				BitSet			
9) Множество зарытых или открытых отрезков					RangeSet RangeMap		
10) Деревья		TreeSet	TreeMap			PatriciaTrie	TreeSortedSet
11) Кеши			LinkedHashMap WeakHashMap		LoadingCache		

VII. Заключение

Я специально не буду делать выводов какая библиотека хуже или лучше, так как во многом это дело вкуса, но в любом случае в альтернативных коллекциях можно найти много полезных коллекций и методов, если знать где искать. Спасибо, за то что дочитали (или долистали) до конца, надеюсь вы сумели найти в этой статье что-нибудь для себя полезное.

Исходные коды всех примеров можно найти на [github'e](#).

Источники, которые использовались для написания статьи:

1. [Обзор java.util.concurrent.* tutorial](#)
2. [Trove library: using primitive collections for performance](#)
3. [Java performance tuning tips](#)
4. [Large HashMap overview](#)
5. [Memory consumption of popular Java data types](#)
6. И, естественно, официальная документация, javadoc и исходные коды всех рассмотренных библиотек

P.P.S. Так же советую посмотреть мой opensource проект [useful-java-links](https://github.com/Vedenin/useful-java-links/tree/master/link-rus) — возможно, наиболее полная коллекция полезных Java библиотек, фреймворков и русскоязычного обучающего видео. Так же есть аналогичная [английская версия](https://github.com/Vedenin/useful-java-links/) этого проекта и начинаю opensource подпроект [Hello world](https://github.com/Vedenin/useful-java-links/tree/master/helloworlds) по подготовке коллекции простых примеров для разных Java библиотек в одном maven проекте (буду благодарен за любую помощь).

▼ Общее оглавление 'Шпаргалок'

1. JPA и Hibernate в вопросах и ответах
2. Триста пятьдесят самых популярных не мобильных Java opensource проектов на github
3. Коллекции в Java (стандартные, guava, apache, trove, gs-collections и другие)
4. Java Stream API
5. Двести пятьдесят русскоязычных обучающих видео докладов и лекций о Java
6. Список полезных ссылок для Java программиста
- 7 Типовые задачи
 - 7.1 Оптимальный путь преобразования InputStream в строку
 - 7.2 Самый производительный способ обхода Map'ы, подсчет количества вхождений подстроки
8. Библиотеки для работы с Json (Gson, Fastjson, LoganSquare, Jackson, JsonPath и другие)

Какие библиотеки альтернативных коллекций вы используете в своей работе?

- ☐ Guava
- ☐ Apache Commons Collections
- ☐ Trove
- ☐ GC-collections
- ☐ Fastutil
- ☐ HPPC
- ☐ Koloboke
- ☐ GlazedLists collections
- ☐ Concurrent Building Blocks
- ☐ Functional Java
- ☐ Primitive Collections for Java
- ☐ Apache Jakarta Commons Primitives
- ☐ Chronicle-Queue
- ☐ Использую свои реализации коллекций
- ☐ Использую только стандартный фреймворк коллекций JDK
- ☐ Другая библиотека (напишите, пожалуйста какая в комментариях)

Проголосовало 366 человек. Воздержался 181 человек.

Только зарегистрированные пользователи могут участвовать в опросе. Войдите, пожалуйста.

Java, коллекции, алгоритмы, производительность, stream, stream api, java 8, примеры кода

↑ +57 ↓

105k

★ 1126

Twitter

VK

F

Telegram

Автор: @vedenin1980

рейтинг

Luxoft 106,62

Сайт

Twitter

Похожие публикации

+25

Шпаргалка Java программиста 4. Java Stream API

116k

★ 727

17

+37

Как работает декомпиляция в .Net или Java на примере .Net

25,4k

★ 249

28

+3

Мастер-класс Адама Бина «Java EE: Архитектура, шаблоны и решения»: отзывы и впечатления участников

5,6k

★ 24

5

Комментарии (39)

roces

28 октября 2015 в 02:16

#

+4

↑

↓

LinkedList — рекомендуется не использовать

https://habrahabr.ru/company/luxoft/blog/256877/

45/49

И почему же LinkedList рекомендуется не использовать?



vedenin1980 28 октября 2015 в 02:39 (комментарий был изменён) # h ↑

+9 ↑ ↓

В целом, потому что

1) он требует в шесть раз больше памяти чем ArrayList (примерно в 4 раза чем ArrayDeque), даже с учетом выделения ArrayList памяти с запасом, все равно по памяти он проигрывает,

2) случаи, когда он будет выигрывать у ArrayList/ArrayDeque встречаются не так часто, так как даже с учетом вставки в середину, итерирование сильно бьет по производительности. То есть он хорош только если надо часто вставлять или удалять данные в середине списка и при этом практически «не двигаться», что довольно редкий случай. На [оф.сайте](#) рекомендую всегда перед выбором Linkedlist сравнить производительность алгоритма по сравнению с ArrayList/ArrayDeque, так как чаще всего выигрыша не будет, даже при вставках внутри листа,

3) если есть возможность использовать Apache Common Collections, то TreeList на основе дерева намного лучше альтернатива для LinkedList, так как и вставка внутрь списка происходит быстро и получение данных по индексу тоже быстрое.

В целом, LinkedList можно использовать в довольно редких случаях, если Apache Common Collections ну никак в проекте использовать нельзя (что я встречал довольно редко).



roces 28 октября 2015 в 13:19 # h ↑

0 ↑ ↓

Я частично согласен с вашими утверждениями, но все же есть ситуации, когда LinkedList стоит использовать и они не так уж редки. Из той же документации:

If you frequently add elements to the beginning of the List or iterate over the List to delete elements from its interior, you should consider using LinkedList

К тому же есть много алгоритмов, и структур данных, которые используют именно связный список.

Поэтому я не стал бы так критично писать о том, что его не рекомендуется использовать. Кто-нибудь начинающий, например, может просто запомнить это и подумать что это что-то устаревшее.



vedenin1980 28 октября 2015 в 13:28 (комментарий был изменён) # h ↑

0 ↑ ↓

Ок, не буду спорить, поправил на «полезен лишь в некоторых редких случаях»



dougrinch 28 октября 2015 в 15:16 # h ↑

+5 ↑ ↓

If you frequently add elements to the beginning of the List

В этом случае гораздо лучше будет ArrayDeque.

Кто-нибудь начинающий, например, может просто запомнить это и подумать что это что-то устаревшее.

Имхо, как раз будет лучше если он так подумает. Т.к. если у него действительно тот единственный процент случаев, где необходим именно LinkedList, то после непродолжительного поиска он его найдет. Во всех же остальных случаях незнание пойдет только на пользу.



ZnW 28 октября 2015 в 03:44 (комментарий был изменён) # h ↑

+3 ↑ ↓

Вы же знакомы с промахами в разных уровнях кеша процессора? LinkedList соединяет данные из (почти всегда) зачастую разных неупорядоченных участков памяти, в итоге получаем постоянные кеш миссы и, как следствие, падение производительности. Остальное уже пояснил второй комментатор. ~~И на самом деле это единственное, что я знаю из недавно прочтенной статьи~~



AlexanderG 29 октября 2015 в 14:38 # h ↑

0 ↑ ↓

В случае JVM имеет ли смысл задумываться о расположении данных в кеше? Я имею в виду, сама машина достаточно эффективно реализована, чтобы можно было это учитывать? У нас ведь очень высокоуровневое управление памятью, и как оно там внутри VM реализовано, неизвестно.



grossws 29 октября 2015 в 14:59 # h ↑

+2 ↑ ↓

Иногда имеет смысл.

Классический пример — умножение матриц. Если одну оставить row-major, а вторую привести к column-major, то по скорости оно почти не отличается от такого же наивного алгоритма на C. Если этого не делать (или тем более использовать вложенные массивы), то падение производительности на порядки. Пример немного синтетический, в production я бы просто взял blas.

Если говорить про реальность — всякие индексы, фильтры Блума и т. п. довольно чувствительны к выравниванию на cache line и порядок доступа. Особенно, при наличии concurrency.




grossws 28 октября 2015 в 03:26 #

+3 ↑ ↓


Внушительная работа, спасибо.

Для меня основными фишками guava стали immutable-коллекции и их билдеры, multimap, а также Collections2, Iterables etc.

 **leventov** 28 октября 2015 в 05:24 (комментарий был изменён) #


+6 ↑ ↓

Если хотите принести пользу сообществу — сделайте большой раздел про fastutil или GS, а про Trove лишь маленькое упоминание, что их **ПЛОХО** использовать в новых проектах. Они *строго* хуже fastutil/GS по любым параметрам (кол-во багов, полнота покрытия интерфейсов, производительность, активность поддержки, и т. д.)

 **vedenin1980** 28 октября 2015 в 13:43 # ↵ ↑

0 ↑ ↓

Хорошо, добавил ваш комментарий в статью в раздел описания trove.

 **leventov** 28 октября 2015 в 05:32 (комментарий был изменён) #

+3 ↑ ↓

Интересно, каким образом в опросник пролезла Chronicle Queue, хотя коллекцией это можно назвать с большой натяжкой. (Кстати, в принципе сейчас считается ошибкой дизайна JCF, что Queue расширяет Collection). Тогда если упоминать, то [JCTools](#).


Если говорить об off-heap «коллекциях», то это отдельная большая тема, надо уже делать обзор Hazelcast/Infinispan/Apache Ignite/Ehcache/Terracota/MapDB/Chronicle и т. д.

 **Throwable** 28 октября 2015 в 13:43 #

+1 ↑ ↓

Поправьте везде GC-collections на GS-collections.

Часто использую коллекции из Guava. Плохо, что code completion для итерации в Idea не понимает новых типов коллекций. Поэтому зачастую быстрее написать все, используя стандартные коллекции из Java.

 **vedenin1980** 28 октября 2015 в 13:53 # ↵ ↑

0 ↑ ↓

О, спасибо, поправил

 **alexzzam** 28 октября 2015 в 19:43 #

+1 ↑ ↓


интерфейсах Query/Dequery (очереди)

Имеется в виду Queue/Deque

 **vedenin1980** 28 октября 2015 в 23:59 # ↵ ↑

0 ↑ ↓

Да, поправил спасибо

 **jorgen** 28 октября 2015 в 23:58 (комментарий был изменён) #

+3 ↑ ↓

Отличный обзор, спасибо!


Сам я в конце-концов пришёл к собственным коллекциям. По многим причинам, но в основном, чтобы мочь писать так:

```
String names = al(new File("/home/user/").listFiles())
    .filter(File::isDirectory)           //only dirs
    .map(File::getName)                  //get name
    .filter(n -> n.startsWith("."))      //only invisible
    .sorted()                           //sorted
    .foldLeft("", (r, n) -> r + ", " + n); //to print fine
```

 **leventov** 29 октября 2015 в 04:47 # ↵ ↑

+2 ↑ ↓

Чем плохо Gs и java 8 streams?

 **jorgen** 29 октября 2015 в 14:59 # ↵ ↑

+1 ↑ ↓

Java 8 streams отпадает, потому что плюс две строчки даже если нужно просто отфильтровать. Gs — его ещё не было. Да и сейчас — мне нужны стандартные java-коллекции с плюшками, а не другие.

 **leventov** 29 октября 2015 в 16:49 # ↵ ↑

+1 ↑ ↓

Эти лишние две строчки — для того, чтобы в конечном счёте делать чаще всего 1-2 прохода по данным, с одним созданием конечной коллекции, а не создавать кучу промежуточного мусора.

 **grossws** 29 октября 2015 в 17:20 # ↵ ↑

0 ↑ ↓

В той же scala такие операции ленивы, и либо далее используется получившиеся ленивые варианты, либо делается toArray/toSeq/toVector, который создаёт итоговую коллекцию.



senia 29 октября 2015 в 22:16 (комментарий был изменён) # h ↑

+2 ↑ ↓

К счастью нет. В scala нет неявных ленивых операций (withFilter не для вызова вручную). Используйте .view.



grossws 30 октября 2015 в 18:05 # h ↑

+1 ↑ ↓

Перепутал. Это итераторы ленивые, что логично. Например

```
val it = Seq(1, 2, 3).toIterator.map { x=> print(x); x }
print("x")
it.toVector
```

выведет x123.



fshp 30 октября 2015 в 02:33 # h ↑

+2 ↑ ↓

Это вы с haskell перепутали. Там как раз по умолчанию всё ленивое, а жадность — по запросу. В scala же всё жадное по умолчанию, лень — по запросу.



jorgen 29 октября 2015 в 17:43 # h ↑

+1 ↑ ↓

Ну, пока мне JProfiler не скажет, я даже переменную не выделяю :) Утрирую, конечно, но иногда десяток лишних копий в профайлере не видны, а иногда — приходится итераторы на индексы заменять.



leventov 29 октября 2015 в 16:56 (комментарий был изменён) # h ↑

+1 ↑ ↓

И, кстати, GS наследует стандартным интерфейсам, ровно так же, как и ваши. Те, которые объект-объект. Прimitives специализации — не наследуют, но у вас их и нет, как я понял. Но, согласен, — тянуть жирный-прежирный GS, если не нужны примитивные — может показаться сомнительным



fshp 29 октября 2015 в 14:12 (комментарий был изменён) # h ↑

+3 ↑ ↓

Вы изобрели Scala. Поздравляю.



jorgen 29 октября 2015 в 15:08 # h ↑

+1 ↑ ↓

Я знаю, спасибо. А ещё Xtend и Kotlin. Но Xtend был ещё сырой и только эклипс. Скала — ~~я могу лекцию прочитать чем она плоха~~ мне не нравится. На Котлин вот посматриваю. Но пока — java+«[мои коллекции](#)» дают мне наибольшую продуктивность.



olegchir 3 ноября 2015 в 10:54 # h ↑

+2 ↑ ↓

лекция была бы интересна



dShell 30 октября 2015 в 09:13 # h ↑

+1 ↑ ↓

Guava: Fluent list + predicates + functions + chain



MercurieVV 30 октября 2015 в 04:32 #

+2 ↑ ↓

Так как на Андроиде нет J8 (и Streams), использую это github.com/aNNiMON/Lightweight-Stream-API



farewell 3 ноября 2015 в 13:48 #

0 ↑ ↓

Долго тупил, почему именно Java 3.



vedenin1980 3 ноября 2015 в 13:51 # h ↑

+1 ↑ ↓

Исправил заголовок, надеюсь стало понятнее.



farewell 3 ноября 2015 в 14:17 # h ↑

0 ↑ ↓

Спасибо! :)



radistao 3 ноября 2015 в 14:49 #

-1 ↑ ↓

Результаты голосования:

51% (125) Guava
50% (121) Использую **только** стандартный фреймворк коллекций JDK
Проголосовало 243 человека

101%... Даже IT-шники не понимают, за что голосуют. Вот и выборы у нас так проводят...



grossws 3 ноября 2015 в 14:51 # h ↑

0 ↑ ↓

Варианта без «только», как можно заметить, нет. При том, что остальные варианты не являются взаимоисключающими.



radistao 3 ноября 2015 в 14:56 # h ↑

0 ↑ ↓

эти 2 варината как раз являются взаимоисключающими: если человек выбрал Guva (да и любой другой), он не может выбрать «только JDK», и наоборот, выбравший «только JDK» не может выбрать «Guva» (или любой другой). Из 243 человек 121 отвечает «только JDK», значит guva могут использовать не более 243-121=122, а выбрало 125.



grossws 3 ноября 2015 в 15:00 # h i

0 ↑ ↓

Ещё раз. Выбрать, например, guava + commons-collection — можно, они не взаимоисключающие, как и любая другая пара (и более) ответов, кроме j.u.c, что нелогично. Предполагаю, что многие решили, что автор опроса некорректно сформулировал пункт про j.u.c и воспринимали его без слова «только».



radistao 3 ноября 2015 в 15:18 # h i

0 ↑ ↓

и воспринимали его без слова «только»

вот именно это я и имел в виду в своем первом комментарии: даже IT-шники, «высокообразованные» люди, не умеют правильно голосовать, и могут выбрать 2 взаимоисключающих пункта при голосовании! Ведь это же очевидно, что это мультиголосование, и сумма всех результатов может быть больше 100%. Но я обращаю внимание именно на эти 2 взаимоисключающие условия.

Только зарегистрированные пользователи могут оставлять комментарии. [Войдите](#), пожалуйста.

Самое читаемое

Разработка

Сейчас Сутки Неделя Месяц

+242 [Как Skype уязвимости чинил](#)

👁 23,2k ★ 98 💬 133

+19 [Улучшение производительности PHP 7](#)

👁 2,3k ★ 25 💬 3

+77 [Здравствуй, дорогой Мегафон](#)

👁 8,9k ★ 15 💬 78

+10 [Прототип RFC HTTP-кодов состояния для ошибок разработчиков \(диапазон 7XX\)](#)

👁 1,4k ★ 4 💬 6

+12 [Компания Google представила набор тестов Wycheproof](#)

👁 1,2k ★ 7 💬 0

Интересные публикации



[Гейзенбаг: Версия 1.0](#) 💬 0

[Компания Google представила набор тестов Wycheproof](#) 💬 0

[Улучшение производительности PHP 7](#) 💬 3

[Защищенный Dell](#) 💬 4

[Преобразование формы представления данных при помощи Excel+PowerQuery](#) 💬 0