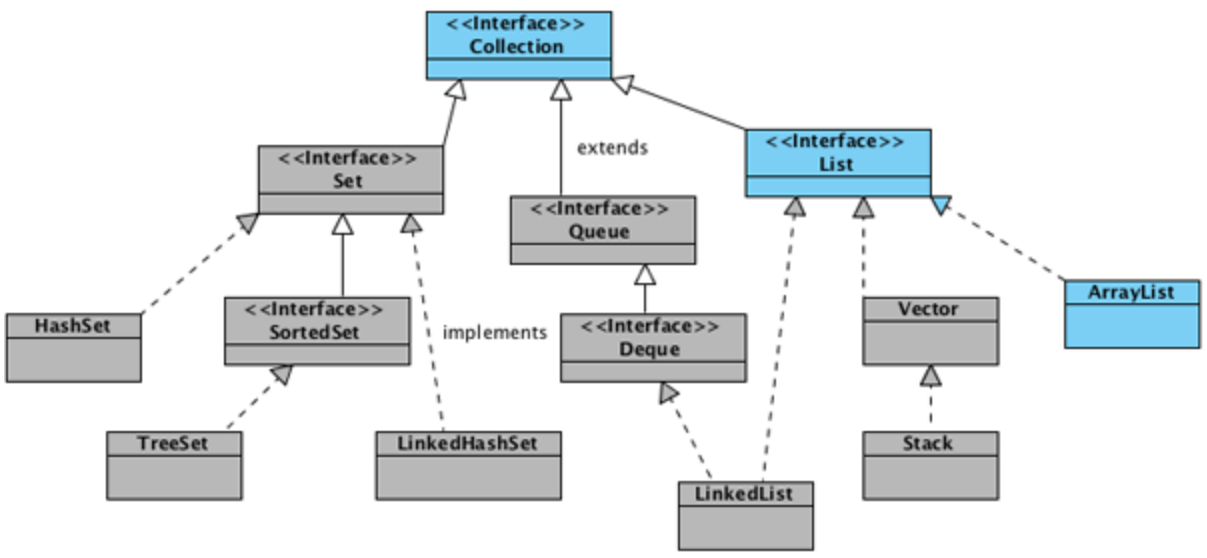
tarzan82 12 сентября 2011 в 17:19

Структуры данных в картинках. ArrayList

Java*

Приветствую вас, хабралюди!

Взбрело мне в голову написать несколько статей, о том как реализованы некоторые структуры данных в Java. Надеюсь, статьи будут полезны визуалам (картинки наше всё), начинающим java-визуалам а также тем кто уже умеет писать new ArrayList(), но слабо представляет что же происходит внутри.



Сегодня поговорим о ArrayList-ах

ArrayList — реализует интерфейс List. Как известно, в Java массивы имеют фиксированную длину, и после того как массив создан, он не может расти или уменьшаться. ArrayList может менять свой размер во время исполнения программы, при этом не обязательно указывать размерность при создании объекта. Элементы ArrayList могут быть абсолютно любых типов в том числе и null.

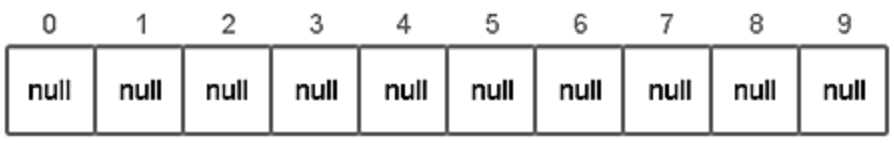
Создание объекта

```
ArrayList<String> list = new ArrayList<String>();
```

Только что созданный объект list, содержит свойства **elementData** и **size**.

Хранилище значений **elementData** есть ни что иное как массив определенного типа (указанного в generic), в нашем случае **String[]**. Если вызывается конструктор без параметров, то по умолчанию будет создан массив из 10-ти элементов типа Object (с приведением к типу, разумеется).

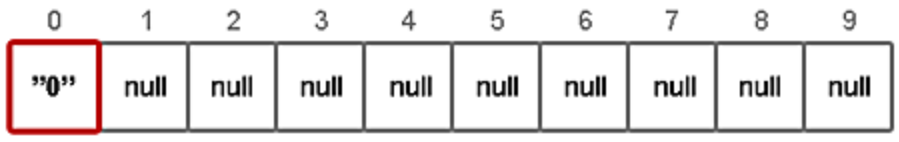
```
elementData = (E[]) new Object[10];
```



Вы можете использовать конструктор **ArrayList(capacity)** и указать свою начальную емкость списка.

Добавление элементов

```
list.add("0");
```



Внутри метода **add(value)** происходят следующие вещи:

1) проверяется, достаточно ли места в массиве для вставки нового элемента;

```
ensureCapacity(size + 1);
```

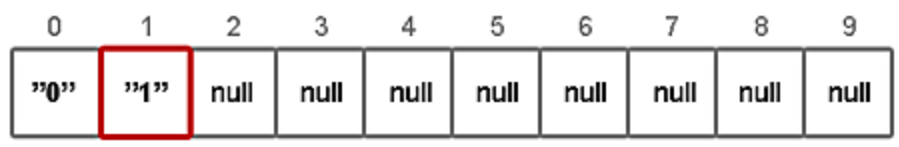
2) добавляется элемент в конец (согласно значению **size**) массива.

```
elementData[size++] = element;
```

Весь метод **ensureCapacity(minCapacity)** рассматривать не будем, остановимся только на паре интересных мест. Если места в массиве не достаточно, новая емкость рассчитывается по формуле **(oldCapacity * 3) / 2 + 1**. Второй момент это копирование элементов. Оно осуществляется с помощью **native** метода **System.arraycopy()**, который написан не на Java.

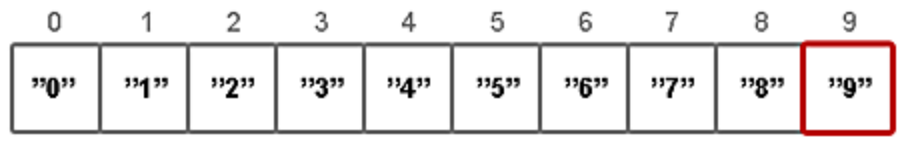
```
// newCapacity - новое значение емкости
elementData = (E[])new Object[newCapacity];

// oldData - временное хранилище текущего массива с данными
System.arraycopy(oldData, 0, elementData, 0, size);
```

Ниже продемонстрирован цикл, поочередно добавляющий 15 элементов:



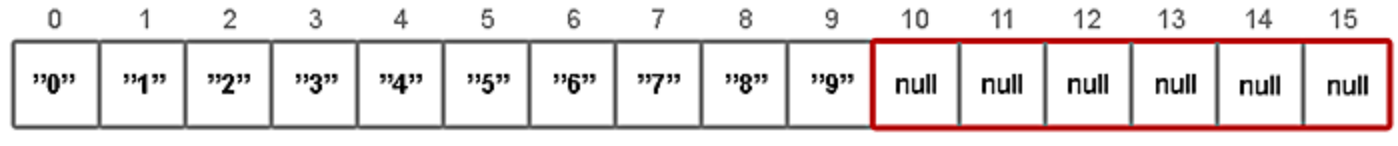
...

```
list.add("9");
```

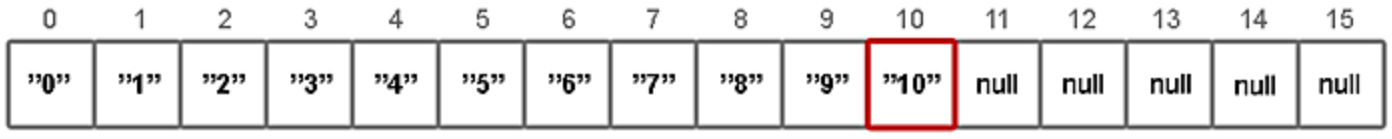



```
list.add("10");
```

При добавлении 11-го элемента, проверка показывает что места в массиве нет. Соответственно создается новый массив и вызывается **System.arraycopy()**.

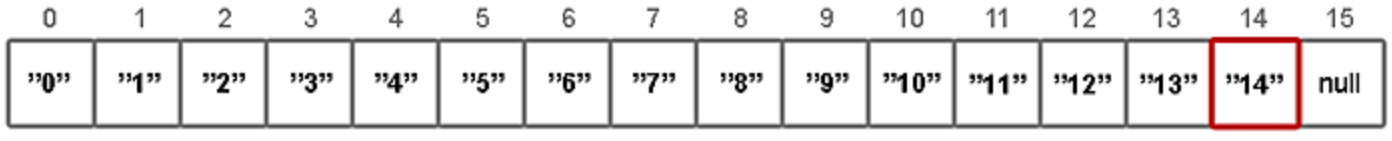


После этого добавление элементов продолжается



...

```
list.add("14");
```



Добавление в «середину» списка

```
list.add(5, "100");
```

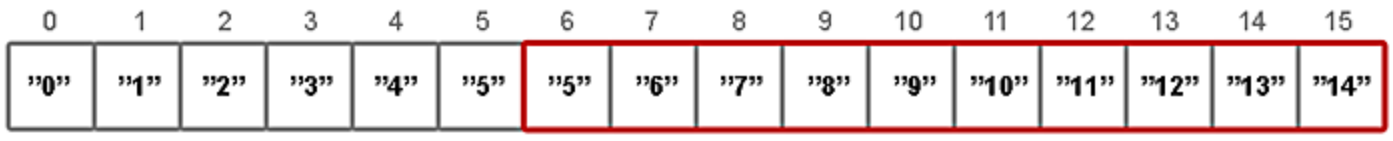
Добавление элемента на позицию с определенным индексом происходит в три этапа:

1) проверяется, достаточно ли места в массиве для вставки нового элемента;

```
ensureCapacity(size+1);
```

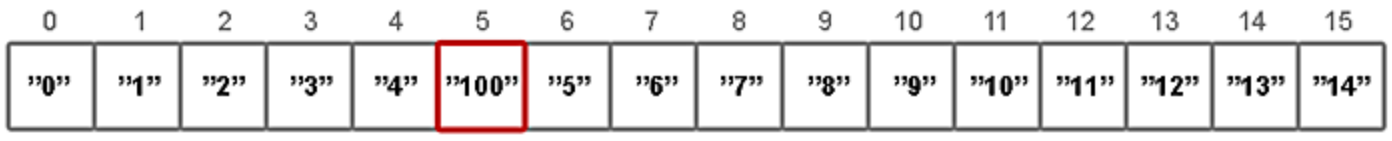
2) подготавливается место для нового элемента с помощью **System.arraycopy()**;

```
System.arraycopy(elementData, index, elementData, index + 1, size - index);
```

3) перезаписывается значение у элемента с указанным индексом.

```
elementData[index] = element;
size++;
```



Как можно догадаться, в случаях, когда происходит вставка элемента по индексу и при этом в вашем массиве нет свободных мест, то вызов **System.arraycopy()** случится дважды: первый в **ensureCapacity()**, второй в самом методе **add(index, value)**, что явно скажется на скорости всей операции добавления.

В случаях, когда в исходный список необходимо добавить другую коллекцию, да еще и в «середину», стоит использовать метод **addAll(index, Collection)**. И хотя, данный метод скорее всего вызовет **System.arraycopy()** три раза, в итоге это будет гораздо быстрее поэлементного добавления.

Удаление элементов

Удалять элементы можно двумя способами:
— по индексу **remove(index)**

— по значению **remove(value)**

С удалением элемента по индексу всё достаточно просто

```
list.remove(5);
```

Сначала определяется какое количество элементов надо скопировать

```
int numMoved = size - index - 1;
```

затем копируем элементы используя **System.arraycopy()**

```
System.arraycopy(elementData, index + 1, elementData, index, numMoved);
```

уменьшаем размер массива и забываем про последний элемент

```
elementData[--size] = null; // Let gc do its work
```

При удалении по значению, в цикле просматриваются все элементы списка, до тех пор пока не будет найдено соответствие. Удален будет лишь первый найденный элемент.

Дополнение 1: Как верно заметил @MikeMirzayanov, при удалении элементов текущая величина capacity не уменьшается, что может привести к своеобразным утечкам памяти. Поэтому не стоит пренебрегать методом **trimToSize()**.

Итоги

— Быстрый доступ к элементам по индексу за время O(1);
— Доступ к элементам по значению за линейное время O(n);
— Медленный, когда вставляются и удаляются элементы из «середины» списка;
— Позволяет хранить любые значения в том числе и null;
— Не синхронизирован.

Ссылки

Исходник ArrayList
Исходник ArrayList из JDK7
Исходники JDK OpenJDK & trade 6 Source Release — Build b23

Пишите в комментариях пожелания/замечания и есть ли смысл продолжать.

Теги: java, структуры данных, ArrayList

Хабы: Java

113 Карма 0 Рейтинг

@tarzan82

Пользователь

Комментарии 66

QtRoS 12.09.2011 в 17:44

Спасибо, было приятно убедится, что все происходит именно так, как и должно было :)

P.S. Местами шаблоны Java накладывают свои ограничения, например нельзя массив типа шаблона создать…

+1 Ответить

spiff 12.09.2011 в 19:19

Что вы имеете ввиду, говоря «массив типа шаблона»? Generic?

0 Ответить

QtRoS 12.09.2011 в 19:27

Я говорю массив типа E создать нельзя, написав new E[len], как собственно и просто объект.

-2 Ответить

Moonrise 12.09.2011 в 19:36

Ну с объектом всё понятно — во время компиляции же неизвестно, какой конструктор необходимо вызвать

+2 Ответить

iximiuz 12.09.2011 в 19:43

Какой конструктор? Если не изменяет память, то при создании списка все элементы инициализируются значением null. А список хранит в себе ссылки на объекты, причем все наследуются от object. Так что хранить в списке можно и пользовательские типы

+1 Ответить

QtRoS 12.09.2011 в 19:46

Когда вы пишите
MyClass[] arr = new MyClass[10];
10 раз вызовется стандартный конструктор класса MyClass. Поэтому нельзя создать объект типа E, неясно что вызывать!

-8 Ответить

edhell 12.09.2011 в 19:48

Видимо, Вы программируете на чём-то отличном от Java. В Java new MyClass[10] создает только массив, объектов класса MyClass он не создаёт.

+7 Ответить

iximiuz 12.09.2011 в 19:50

Просто речь в статье шла об ArrayList, я позволил себе предположить, что вы в своем комментарии именно его сократили до «массив». В остальном же полностью с вами согласен

0 Ответить

Moonrise 12.09.2011 в 19:50

Нет, элементы массива в Java при создании не создаются, инициализируются в null. Здесь скорее проблема в том, что тип массива также не может быть определён во время компиляции. В отличие от дженериков, которые чисто языковая фича, типы Object[] и String[] различны

0 Ответить

QtRoS 12.09.2011 в 20:03

Вы правы, товарищи, к вечеру меня потянуло путать с C++, глубоко извиняюсь за то, что привел в заблуждение!

+3 Ответить

romik 12.09.2011 в 19:27

> elementData = (E[]) new Object[10];

Никакого приведения типа там нет и быть не может, массив так и остаётся Object[].

+1 Ответить

EgorK 12.09.2011 в 19:53

Там его нет, но быть может
Например вот такой идиотский класс:

```
 public class Holder {
private T[] array;
public Holder() {
//noinspection unchecked
this.array = (T[]) new Object[1];
}

public void set(T element) {
array[0] = element;
}

public T get() {
return array[0];
}

public static void main(String[] args) {
Holder holder = new Holder();
holder.set("Hi");
System.out.println(holder.get());
}
}
```

вполне себе работает и выдает что нужно

-1 Ответить

romik 12.09.2011 в 20:10

Ваш *//noinspection unchecked* как раз и намекает на то, что в реальности никакого приведения типов не происходит. А если бы оно происходило то как раз бы ничего не работало:

```
 String[] d = (String[]) new Object[1];
--> java.lang.ClassCastException: [Ljava.lang.Object; cannot be cast to
[Ljava.lang.String;
```

0 Ответить

EgorK 12.09.2011 в 20:17

В реальности — не происходит, конечно. Однако жить и работать с получившимся T[] гораздо проще, чем с Object[] — потому не совсем понятно, почему этого не сделали в ArrayList и прочих Collection

-1 Ответить

Moonrise 12.09.2011 в 20:24

Наверное потому, что они появились раньше, чем дженерики, а потом переписывать уже никто не хотел.

0 Ответить

EgorK 12.09.2011 в 20:30

Очевидно, что с появлением generic'ов они переписывали этот код. Соответственно, либо приведение каждый раз при создании массива, либо каждый раз при обращении. По-моему первый путь экономнее

0 Ответить

barker 12.09.2011 в 21:07

> Соответственно, либо приведение каждый раз при создании массива, либо каждый раз при обращении. По-моему первый путь экономнее

Какое ещё приведение? Ни в первом ни во втором случае нет никакого приведения. И вообще никакого приведения не бывает с дженериками. В реальности array останется массивом Object и в том числе в байткоде. Вы как-то превратно понимаете что такое java-дженерики.

+1 Ответить

EgorK 12.09.2011 в 21:09

Я все прекрасно понимаю. Есть такая вещь — простота редактирования кода. Я имею ввиду «виртуальное» приведение — то, которое нужно компилятору

0 Ответить

barker 12.09.2011 в 21:16

А, ок. Да, в случае исходников тут всё понятно — вроде как можно и правда переписать Object[] на T[], но резона, видимо, не было) Да и разницы было бы не много — в некоторых геттерах убралось бы «виртуальное приведение» (T[]). С точки зрения внешнего интерфейса и производительности всё останется идентичным.

+1 Ответить

romik 12.09.2011 в 23:14

Наверное потому, что в Sun старались писать хороший код? По-моему, код, в котором без особой на то нужды игнорируются предупреждения компилятора о типобезопасности generic-ов, не слишком хорош.

0 Ответить

EgorK 12.09.2011 в 23:16

A return (T)elementData[index]; по вашему его не генерирует? Невозможно на Java написать генерик коллекцию на основе массива без type-safe warning'a к сожалению

0 Ответить

romik 12.09.2011 в 23:57

Да, сглупил. Но всё же я остаюсь при своём мнении о «хорошести» кода. Альтернативный аргумент: операция (T) elementData[index] осталась бы корректна, если бы вместо T был реальный тип, а операция (T[]) new Object[capacity] возможна **только** для generic-типов. По-сути, когда мы пишем приводение к T[], мы на самом деле рассчитываем на то что эта операция не сработает, что несколько странно. В чуть-более сложной ситуации типа class MyCollection<T extends Number> это допущение сразу перестаёт работать:

```
class A<T extends Number>:
    (T[]) new Object[1];
    --> java.lang.ClassCastException: [Ljava.lang.Object; cannot be cast to
[Ljava.lang.Number;
```

Кстати, написать коллекцию без warning-ов можно, достаточно передать в конструктор Class<T>. Только это никому не нужно.

0 Ответить

edhell 12.09.2011 в 19:32

Проще говоря, ArrayList ведёт себя как обычный массив. Единственно стоит учитывать, что при переполнении массив удваивается в размере.

Чуть интереснее с TreeMap/TreeSet (красно-чёрные деревья) и HashMap/HashSet (хэш-таблица в виде массива списков).

0 Ответить

edhell 12.09.2011 в 19:42

поправка: не в два раза, а в полтора: int newCapacity = (oldCapacity * 3)/2 + 1

0 Ответить

spiff 12.09.2011 в 20:29

Дада. Поэтому, если вы заранее знаете сколько будете хранить в контейнере — лучше сразу указать его размер в конструкторе, тем самым избежать копирование (arraycopy), которое как известно хоть и вылизано до последней строчки, но всеравно жутко дорогое.

+2 Ответить

TimID 12.09.2011 в 20:19

Вам бы книги писать по программированию «для чайников»
Очень все красиво и наглядно.

+6 Ответить

tarzan82 12.09.2011 в 21:22

Спасибо, рад что понравилось

+4 Ответить

spiff 12.09.2011 в 20:37

Мне кажется надо продолжать. Особенно мне понравились итоги — краткие выводы: стоимость операций. Добавить еще и мотивацию, вроде: «Используй ArrayList везде, где невозможно обойтись массивом», вообще бы цены не было.

+4 Ответить

tarzan82 12.09.2011 в 21:27

Спасибо, постараюсь учесть в будущих статьях

+1 Ответить

Optik 12.09.2011 в 22:48

> «Используй ArrayList везде, где невозможно обойтись массивом»

По-моему, это не совсем верно. Поправьте, если ошибаюсь, но часто бывают ситуации, когда предпочтительней к примеру LinkedList.
И еще, мне кажется, говоря о коллекциях, стоит упомянуть итераторы.

+2 Ответить

Akvel 13.09.2011 в 06:49

Судя по недавнему обзору Сравнение потребления памяти у разных структур хранения данных LinkedList уж больно прожорлив по памяти

0 Ответить

Optik 13.09.2011 в 07:45

У LinkedList есть преимущество по скорости при наполнении элементами. При этом проход итератором по списку, подозреваю, будет одинаков.

Не ожидал, что LinkedList так много жрет памяти. Но при относительно больших по размеру объектах разница нивелируется, а учитывая, что ArrayList может выделить под новый элемент в полтора раза больше памяти, тогда он и «пожирней» может оказаться.

-1 Ответить

Regis 13.09.2011 в 09:04

> У LinkedList есть преимущество по скорости при наполнении элементами.

В большинстве типичных ситуаций это неверно. Тесты показывают, что LinkedList действительно эффективен становится только когда вам нужны операции вставки в середину/удаления из середины (что вполне ожидаемо). Если же вам это не нужно, то лучше использовать ArrayList. Даже если в конкретной ситауции они будет не лучше LL, то будет отставать совсем незначительно.

+3 Ответить

Optik 13.09.2011 в 11:01

Ясно. Можно линк на тесты?

0 Ответить

Regis 13.09.2011 в 17:10

Не совсем те тесты, но сойдет:
http://jdevel.wordpress.com/2011/03/01/arraylist-vs-linkedlist/

0 Ответить

eugenius_nsk 13.09.2011 в 07:41

> Добавить еще и мотивацию, вроде: «Используй ArrayList везде, где невозможно обойтись массивом», вообще бы цены не было.

Скорее так — «Если не *требуется* вставка/удаление в середину коллекции — всегда использовать ArrayList. А если вставка/удаление в середину необходимы — нужно профилировать и сравнивать производительность с LinkedList.»

0 Ответить

12.09.2011 в 20:45

НЛО прилетело и опубликовало эту надпись здесь

0 Ответить

barker 12.09.2011 в 21:04

Вы же просто общедоступный исходник прочли по строкам.

+8 Ответить

12.09.2011 в 21:53

НЛО прилетело и опубликовало эту надпись здесь

0 Ответить

tarzan82 12.09.2011 в 22:54

Хороших материалов по расходам памяти пока не встречал. Если что-то накопаю, обязательно укажу в след. статьях.

+1 Ответить

Akvel 13.09.2011 в 06:50

Было уже про память тут

0 Ответить

MikeMirzayanov 12.09.2011 в 22:15

Наверное, стоит добавить, что при удалении элемента (remove) текущая величина capacity не уменьшается. То есть если натолкать в ArrayList методами add() 100500 чисел, то его capacity после этого будет не меньше 100500 и останется таким даже если для всех элементов сделать remove(). Это может привести к своеобразным утечкам памяти. Иногда может оказаться полезным метод trimToSize() или запись вида «a = new ArrayList<T>()» для освобождения старого буфера для ссылок в «a».

Еще можно отметить, что если вы создаете ArrayList и знаете, что положите в него не менее n элементов, то немного эффективнее будет создать его как a = new ArraysList<Integer>(n), чтобы буфер не перераспределялся log n раз.

+4 Ответить

tarzan82 13.09.2011 в 00:13

Спасибо, добавил ваше замечание в топик

+2 Ответить

rauch 12.09.2011 в 22:33

А как же Navigable*(Set|Map) в иерархии классов на первой картинке?

+2 Ответить

Optik 12.09.2011 в 23:05

Разве карты относятся к коллекциям?

-2 Ответить

eugenius_nsk 13.09.2011 в 07:45

NavigableSet — да, упущен. А вот NavigableMap (как и SortedMap и, собственно, Map) — не являются Collection-ами.

+2 Ответить

barker 13.09.2011 в 09:10

Не являются коллекциями — значит их реализации в стандартной библиотеке Java не имплементируют Collection<E>? Ну тогда да, но это странное определение. Конечно же, все Map являются коллекциями. Просто их ветка на этом рисунке располагалась бы рядом. Просто почему-то их проигнорировали :(

+1 Ответить

Optik 13.09.2011 в 11:07

> Note also that the hierarchy consists of two distinct trees — a Map is not a true Collection.

download.oracle.com/javase/tutorial/collections/interfaces/index.html

Или я чего то не понимаю? Вы объясните. Я java изучаю недавно, так что без эмоций прошу.

+1 Ответить

12.09.2011 в 23:27

НЛО прилетело и опубликовало эту надпись здесь

0 Ответить

Lof 13.09.2011 в 10:53

Наследование коллекций в Java, то что я всегда не понимал…

0 Ответить

Throwable 13.09.2011 в 11:22

Недостаток ArrayList в том, что он оперирует объектами, и для примитивов его использовать очень неэффективно, хотя в реальной жизни это часто требуется. Вот пара библиотек, которые оперируют коллекциями из примитивов:

trove.starlight-systems.com/
pcj.sourceforge.net/

0 Ответить

corristo 16.09.2011 в 10:20

это принципиальный недостаток generic'ов в яве

0 Ответить

ice9 13.09.2011 в 11:27

Если уж приводите ссылку на сорцы, почему бы не ссылаться на свежую версию и в нормальном месте:
hg.openjdk.java.net/jdk7/jdk7/jdk/file/9b8c96f96a0f/src/share/classes/java/util/ArrayList.java

+1 Ответить

tarzan82 13.09.2011 в 12:12

Я указал исходник по которому делал описание
За ссылку спасибо, добавил в топик

+1 Ответить

jakobz 13.09.2011 в 11:49

>Элементы ArrayList могут быть абсолютно любых типов в том числе и null.
И Unboxed-типы тоже? int, например?

0 Ответить

tarzan82 13.09.2011 в 12:05

Чуть выше @Throwable написал — ArrayList оперирует объектами.
Добавлять вы можете любой тип, другой вопрос как он их хранит.

0 Ответить

bear11 13.09.2011 в 13:09

Есть желание сериализировать и десериализировать ArrayList.
Можно ли его тупо сбрасывать в поток путем
out.writeObject(massiv);
и восстанавливать как
massiv=(ArrayList)in.readObject()?

0 Ответить

Stocker 13.09.2011 в 14:17

Можно, если ваши объекты в списке Serializable, т.е. если мы можете сериализовать ваш объект YourOboject, то можете спокойно кидать в out.writeObject(ArrayList), а потом List massiv = (List)in.readObject().

Другой вопрос в производительности. Я сам не специалист, но есть ощущение что она очень даже хромает. Т.е. у меня большая коллекция таким образом грузится как — то уж очень долго… Думаю переписывать, чтобы вытягивать объекты по очереди и склыдвать уже в новую коллекцию, надеюсь будет быстрее, а будет ли и стоит ли не знаю… Я это, на хлеб этим не зарабатываю если что, тапками не кидайтесь :)

0 Ответить

Stocker 13.09.2011 в 14:17

парсер съел скобки с генериками с YourObject

0 Ответить

bear11 13.09.2011 в 14:35

У меня Array небольшой — штук 20 обьектов и картинок. Производительность посмотрю. Спасибо, подумаю над альтернативой.

0 Ответить

Stocker 13.09.2011 в 14:19

Большое спасибо! Пишите ещё! Это как раз тот случай, когда знать хочется и надо и полезно, но не настолько чтобы самому копаться в исходниках, а Ваша статья как раз то что нужно и по объёму и по

уровню детализации :)

0 Ответить

Andrianovich 09.01.2012 в 14:55

Великолепная статья. Ваш подход к изложению материала удобен.
(Зачем нужен, плюсы, минусы и т.д.) Кратко, лаконично.

0 Ответить

habrahabrovec 09.04.2019 в 21:45

> Если места в массиве не достаточно, новая емкость рассчитывается по формуле (oldCapacity * 3) / 2 + 1

Где вы это взяли? В методе grow() вот что написано:

```
private void grow(int minCapacity) {
        // overflow-conscious code
        int oldCapacity = elementData.length;
        int newCapacity = oldCapacity + (oldCapacity >> 1);
        if (newCapacity - minCapacity < 0)
            newCapacity = minCapacity;
        if (newCapacity - MAX_ARRAY_SIZE > 0)
            newCapacity = hugeCapacity(minCapacity);
        // minCapacity is usually close to size, so this is a win:
        elementData = Arrays.copyOf(elementData, newCapacity);
    }
```

int newCapacity = oldCapacity + (oldCapacity >> 1);
То есть oldCapacity = oldCapacity + (oldCapacity / 2);
Пример: если вместимость была 10, то станет 15 (10 + (10/2) = 15).

0 Ответить

habrahabrovec 09.04.2019 в 23:54

И я проверил только что с помощью рефлексии, именно так это и работает. 10 —> 15 —> 22, и так далее.

0 Ответить

tarzan82 10.04.2019 в 10:58

@habrahabrovec Не забывайте, что статье почти 8 лет. В Java 6 реализация выглядела именно так ка описано в статье, посмотреть можно тут

+2 Ответить

habrahabrovec 12.04.2019 в 02:40

спасибо, теперь понятно.

0 Ответить

Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.

ПОХОЖИЕ ПУБЛИКАЦИИ

2 января в 12:00

LJV: Чему нас может научить визуализация структур данных в Java

+85 23K 236 11 +11

4 сентября 2021 в 16:46

Как снизить зависимость кода от структуры данных?

+2 8.9K 59 41 +41

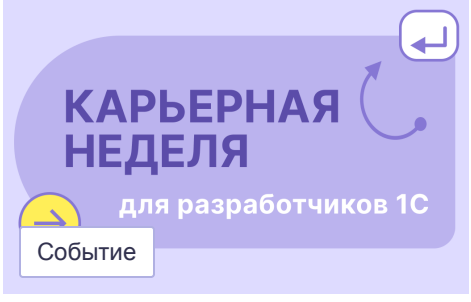
18 июля 2021 в 16:55

Две открытые библиотеки для обучения байесовских сетей и идентификации структуры данных

+6 2.3K 48 1 +1

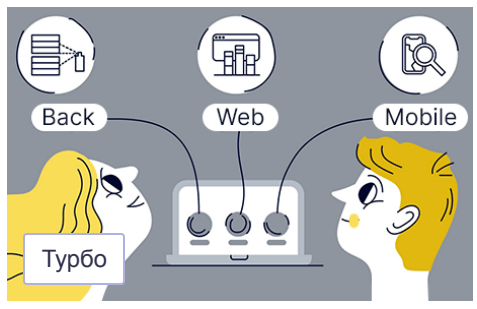
МИНУТОЧКУ ВНИМАНИЯ

Разместить



Карьерная неделя для разработчиков 1С в телеграме

Тетрис на стероидах: тестируем War Robots на Steam Deck



Как проходит интервью QA-инженеров в Тинькофф

ВАКАНСИИ

Java разработчик
от 130 000 · Докзилла · Можно удаленно

Администратор баз данных (Hadoop Администратор)
от 300 000 до 350 000 · АО «ГНИВЦ» · Можно удаленно

Senior Software Developer, Database Engine (Java or C++)
от 400 000 · Querify Labs · Можно удаленно

Lead Software Architect, Database Engine (Java or C++)
от 550 000 · Querify Labs · Можно удаленно

Ведущий инженер-программист в центр роботехники Сбера
от 280 000 до 300 000 · Сбер · Москва

Больше вакансий на Хабр Карьере

ЛУЧШИЕ ПУБЛИКАЦИИ ЗА СУТКИ

вчера в 20:49

Что не так с ДЭГ Москвы на этот раз?

+261 26K 33 98 +98

вчера в 23:33

Смерть Mozilla — это смерть открытого Интернета

+88 23K 36 213 +213

вчера в 16:03

Бифуркация (фантастический рассказ)

+22 3K 12 15 +15

сегодня в 07:06

ЦБ хочет компенсировать страдания российских инвесторов за счет «недружественных» нерезидентов

+21 3.9K 5 3 +3

сегодня в 00:19

Гайд по межсетевому экранированию (nftables)

+17 4.5K 100 2 +2

Хотим узнать, что вы думаете о своём месте работы (анонимно)

Опрос

ЧИТАЮТ СЕЙЧАС

Смерть Mozilla — это смерть открытого Интернета

23K 213 +213

Что не так с ДЭГ Москвы на этот раз?

26K 98 +98

Активность найма на IT-рынке в августе 2022

19K 15 +15

Крякнул софт? Суши сухари

27K 132 +132

Американские компании начали убирать кнопки Facebook** для авторизации со своих сайтов

5.7K 7 +7

Финтех и геймдев: где зеленей трава, а нагрузка больше

Подкаст

РАБОТА

Java разработчик
425 вакансий

Все вакансии

Вернуться на старую версию