10 примеров использования ArrayList в Java – Tutorial

Декабрь 1, 2018

ArrayList в Java является наиболее часто используемым классом коллекции после HashMap . Java ArrayList представляет собой автоматически масштабируемый массив и используется вместо массива. Поскольку мы не можем изменить размер массива после его создания, мы предпочитаем использовать ArrayList в Java, который автоматически изменяет размер при заполнении. **ArrayList в Java реализует интерфейс List и разрешает null** . ArrayList также поддерживает порядок вставки элементов и допускает дублирование, противоположное любой реализации Set, которая не допускает дублирование.

ArrayList поддерживает итераторы и ListIterator для итерации, но рекомендуется использовать `ListIterator` поскольку он позволяет программисту `ListIterator` список в любом направлении, изменять список во время итерации и получать текущую позицию итератора в списке. Но при использовании ListIterator вам нужно быть немного осторожнее, потому что ListIterator не имеет текущего элемента; его позиция курсора всегда находится между элементом, который будет возвращен вызовом `previous()` и элементом, который будет возвращен вызовом `next()` . В этом *руководстве по Java ArrayList* мы увидим, как создавать *Java ArrayList* и выполнять различные операции над Java ArrayList. Этот класс коллекции также является любимым во многих основных Java-интервью с такими вопросами, как Разница между ArrayList и Vector или LinkedList против ArrayList .

ArrayList был изменен в Java 5 (Tiger) для поддержки Generics, что делает **Java ArrayList** еще более мощным из-за повышенной безопасности типов. До Java5, так как не было обобщений, не проводилась проверка типов во время компиляции, что означает, что есть шанс сохранить другой тип элемента в ArrayList, который предназначен для чего-то и в конечном итоге приводит к **ClassCastException** во время выполнения.

С помощью дженериков вы можете создать Java ArrayList, который принимает только тип объекта, указанный во время создания, и приводит к ошибке компиляции, если кто-то пытается вставить любой другой объект в ArrayList в Java; например, если вы создаете объект ArrayList из String, вы не можете хранить в нем Integer, поскольку метод add () объекта ArrayList будет проверять тип перед добавлением объекта в ArrayList в Java в отличие от метода `add()` в Java 1.4, который принимает любой объект.

Java ArrayList с обобщениями в JDK 1.5

Также важно помнить, что **ArrayList не синхронизирован** и не должен использоваться несколькими потоками. Если несколько потоков обращаются к экземпляру Java ArrayList одновременно, и хотя бы один из потоков структурно изменяет список, он должен быть синхронизирован извне. (Согласно Java-документу структурная модификация – это любая операция, которая добавляет или удаляет один или несколько элементов или явно изменяет размеры резервного массива; простая установка значения элемента не является структурной модификацией.)

Обычно это достигается путем синхронизации с некоторым объектом, который естественным образом инкапсулирует список. Если такого объекта не существует, список следует «обернуть» с помощью метода `Collections.synchronizedList()` . Рекомендуется синхронизировать список во время создания, чтобы избежать случайного несинхронизированного доступа к списку. Другим лучшим вариантом является использование `CopyOnWriteArrayList` который добавлен из Java 5 и оптимизирован для одновременного многократного чтения. В `CopyOnWriteArrayList` все мутативные операции (добавление, установка и т. Д.) Реализуются путем создания новой копии базового массива, и поэтому он называется « `CopyOnWrite` ».

Пример ArrayList в Java

Давайте посмотрим **несколько примеров создания ArrayList в Java** и их использования. Я попытался предоставить как можно больше примеров, чтобы проиллюстрировать различные возможные операции в Java ArrayList. Пожалуйста, дайте мне знать, если вам нужны другие примеры Java ArrayList, и я добавлю их здесь.

Java ArrayList Пример 1: Как создать ArrayList

Вы можете использовать **ArrayList в Java с Generics или без него,** оба допускаются универсальными версиями, рекомендуется из-за повышенной безопасности типов.

В этом примере мы создадим ArrayList из String. Этот ArrayList разрешит только String и выдаст ошибку компиляции, если мы попытаемся использовать любой другой объект, кроме String. Если вы заметили, что вам нужно указать тип справа и слева от выражения, в Java 1.7, если вы используете оператор diamond, угловая скобка, вам нужно указать только слева. Это может сэкономить много места, если вы определяете ArrayList для вложенных типов.

```
ArrayList<String> stringList = new ArrayList<String> ; // Generic ArrayList to store only String
ArrayList<String> stringList = new ArrayList<>(); // Using Diamond operator from Java 1.7
```

Java ArrayList Пример 2: Как добавить элемент в ArrayList

Вы можете добавить элементы в ArrayList, вызвав метод add (). Поскольку мы используем Generics, а это ArrayList из String, вторая строка приведет к ошибке компиляции, поскольку этот ArrayList будет разрешать только элементы String.


```
stringList.add("Item"); //no error because we are storing String
stringList.add(new Integer(2)); //compilation error
```

Java ArrayList Пример 3: Как найти размер ArrayList

Размер ArrayList в Java – это общее количество элементов, хранящихся в настоящее время в ArrayList. Вы можете легко найти несколько элементов в ArrayList, вызвав для него метод size (). Помните, что это может отличаться от длины массива, который поддерживает ArrayList. На самом деле резервный массив всегда имеет большую длину, чем размер ArrayList, поэтому он может хранить больше элементов.

```
int size = stringList.size();
```

Java ArrayList Пример 4: Проверка индекса элемента в Java ArrayList

Вы можете использовать `the indexOf()` ArrayList, чтобы узнать индекс определенного объекта. Когда вы используете этот метод, ArrayList внутренне использует метод equals () для поиска объекта, поэтому убедитесь, что ваш элемент реализует `equals()` и `hashCode()` или иначе будет использоваться реализация по умолчанию класса Object, которая сравнивает объект на основе расположения в памяти.


```
int index = stringList.indexOf("Item"); //location of Item object in List
```

Как получить элемент из ArrayList в цикле

Много раз нам нужно **пройти по ArrayList** и выполнить некоторые операции с каждым найденным элементом. Вот два способа сделать это без использования Iterator. Мы увидим использование Iterator в следующем разделе.

```
for (int i = 0; i < stringList.size(); i++)
   String item = stringList.get(i);
   System.out.println("Item " + i + " : " + item);
}

From Java5 onwards you can use foreach loop as well

for(String item: stringList){
System.out.println("retrieved element: " + item);
}
```

Как искать в ArrayList элемент?

Иногда нам нужно *проверить, существует ли элемент в ArrayList в Java* или нет, для этого мы можем использовать метод `contains()` Java. Метод `contains()` принимает тип объекта, определенный при создании ArrayList, и возвращает true, если этот список содержит указанный элемент. Кроме того, вы также можете использовать метод `Collections.binarySearch()` чтобы увидеть, присутствует ли объект в List или нет. `ArrayList` , `Vector` , `CopyOnWriteArrayList` и `Stack implements` интерфейс `RandomAccess` , их можно использовать для выполнения двоичного поиска. Чтобы увидеть, какой подход лучше, см. Эту статью.

Как проверить, пуст ли ArrayList в Java

Мы можем использовать метод `isEmpty()` ArrayList, чтобы проверить, является ли он пустым. Метод `isEmpty()` возвращает true, если этот ArrayList не содержит элементов. Вы также можете использовать метод `size()` для List, чтобы проверить, пуст ли List или нет. Если возвращаемый размер равен нулю, ArrayList пуст.

```
boolean result = stringList.isEmpty(); //isEmpty() will return true if List is empty

if(stringList.size() == 0){
   System.out.println("ArrayList is empty");
}
```

Как удалить элемент из ArrayList

Есть два способа **удалить любые элементы из ArrayList в Java** . Вы можете удалить элемент на основе его индекса или предоставив сам объект. Метод `remove` (int index) и метод `remove(Object o)` используется для удаления любого элемента из ArrayList в Java. Поскольку ArrayList допускает дублирование, стоит отметить, что `remove(Object o)` удаляет первое вхождение указанного элемента из этого списка, если он присутствует. В приведенном ниже коде первый вызов удалит первый элемент из ArrayList, а второй вызов удалит первое вхождение элемента из ArrayList в Java.

```
stringList.remove(0);
stringList.remove(item);
```

Для более подробного обсуждения правильного способа удаления элемента из ArrayList, пожалуйста, проверьте этот учебник .

Копирование данных из одного ArrayList в другой ArrayList

Много раз вам нужно **создать копию ArrayList** для этой цели вы можете использовать `addAll(Collection c)` метод ArrayList в Java для копирования всех элементов из ArrayList в другой ArrayList. Код ниже добавит все элементы `stringList` во вновь созданный `copyOfStringList` .

```
ArrayList<String> copyOfStringList = new ArrayList<String>();
copyOfStringList.addAll(stringList);
```

Как заменить элемент по определенному индексу в ArrayList?

Вы можете использовать метод `set` (`int index` , `E element`) Java ArrayList для замены любого элемента из определенного индекса. `stringList` ниже код заменит первый элемент `stringList` с « `Item` » на « `Item2` «.

```
stringList.set(0,"Item2");
```

Как удалить все элементы из ArrayList?

ArrayList в Java предоставляет метод `clear()` который удаляет все элементы из этого списка. Код ниже `stringList` все элементы из нашего `stringList` и сделает список пустым. **Вы можете повторно использовать Java ArrayList** после его очистки.

```
stingList.clear();
```

Как конвертировать ArrayList в Array в Java

ArrayList предоставляет вам возможность *вернуть массив из вашего ArrayList* . Вы можете использовать метод `toArray(T[] a)` возвращающий массив, содержащий все элементы в этом списке в правильной последовательности (от первого до последнего элемента). « `a` » – массив, в котором должны храниться элементы списка, если он достаточно большой; в противном случае для этой цели выделяется новый массив того же типа времени выполнения.

```
String[] itemArray = new String[stringList.size()];
String[] returnedArray = stringList.toArray(itemArray);
```

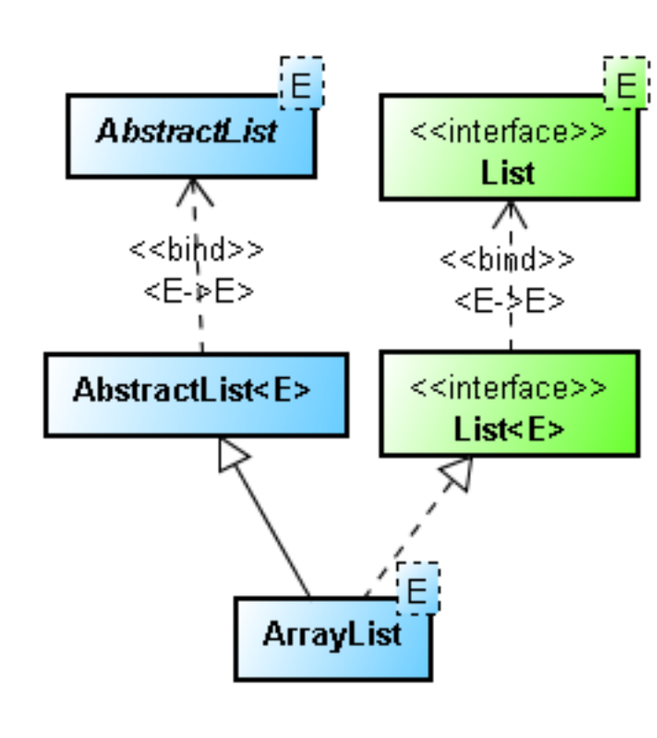
Если вы хотите преобразовать ArrayList обратно в Array, то посмотрите 3 способа преобразования массива в ArrayList в Java

Как синхронизировать ArrayList в Java?

Иногда вам нужно синхронизировать ваш ArrayList в Java, чтобы сделать его доступным для нескольких потоков, для этого вы можете использовать служебный класс Collections, как показано ниже.

```
List list = Collections.synchronizedList(new ArrayList(...));
```

Хотя есть и другие варианты, например, если вам нужен синхронизированный список, вы также можете использовать `CopyOnWriteArrayList` , который является параллельным списком, добавленным в Java 1.5, и работает лучше, чем синхронизированный ArayList, если чтение превосходит запись. Вы также можете просмотреть этот учебник, чтобы узнать больше о том, как синхронизировать ArrayList в Java?



Как создать ArrayList из массива в Java?

ArrayList в Java удивителен, вы можете создать даже ArrayList, полный вашего элемента, из уже существующего массива. Для этого вам нужно использовать метод `Arrays.asList(T... a)` который возвращает список фиксированного размера, подкрепленный указанным массивом.

```
ArrayList stringList = Arrays.asList(new String[]{"One", "Two", "Three"); //this is not read only List you can still update value of existing elements
```

Как перебрать ArrayList в Java?

Вы можете использовать Iterator или ListIterator для **обхода ArrayList** . ListIterator позволит вам перемещаться в обоих направлениях, в то время как Iterator и ListIterator позволят вам *удалять элементы из ArrayList в Java при обходе* .

```
Iterator itr = stringList.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}

ListIterator listItr = stringList.listIterator();
while(listItr.hasNext()){
System.out.println(itr.next());
}
```

посмотрите, как зациклить ArrayList в Java, чтобы найти несколько альтернативных способов обхода List в Java.

Как отсортировать ArrayList в Java?

Ты можешь использовать
Метод `Collections.sort(List list)` для сортировки ArrayList Java в естественном порядке, определенном интерфейсом `Comparable` и может использовать метод `Collections.sort(List list, Comparator c)` для сортировки ArrayList Java на основе пользовательского Comparator. Вы также можете увидеть этот пост, чтобы отсортировать ArrayList в порядке убывания в Java

Как конвертировать ArrayList в HashSet в Java

Большая часть класса Collection предоставляет конструктор, который принимает объект Collection в качестве аргумента. Что можно использовать для копирования всех элементов одной коллекции в другую? `HashSet` также предоставляет такие конструкторы, которые можно использовать для копирования всего объекта из ArrayList в `HashSet` . Но будьте осторожны, поскольку `HashSet` не допускает дублирования, некоторые объекты не будут включены, что приведет к меньшему количеству объектов. См. Как преобразовать ArrayList в HashSet в Java для пошагового примера.

Советы по использованию ArrayList в Java

ArrayList не является синхронизированной коллекцией, поэтому его нельзя использовать между несколькими потоками одновременно. Если вы хотите использовать `ArrayList` как структуру данных в многопоточной среде, то вам нужно либо использовать новый `CopyonWriteArrayList` либо использовать `Collections.synchronizedList()` для создания синхронизированного списка. Бывший является частью пакета одновременного сбора и гораздо более масштабируем, чем второй, но полезен только тогда, когда много читателей и мало записей. Поскольку новая копия `ArrayList` создается каждый раз, когда происходит запись, она может быть излишней, если используется в среде с интенсивной записью. Второй вариант - это строго синхронизированная коллекция, очень похожая на `Vector` или `Hashtable` , но она не масштабируется, потому что, как только число потоков резко увеличивается, конфликт становится огромной проблемой.
`CopyOnWriteArrayList` рекомендуется для параллельной многопоточной среды, поскольку он оптимизирован для нескольких одновременных операций чтения и создает копию для операции записи. Это было добавлено в Tiger, он же JDK 1.5. Он является частью пакета `java.util.concurrent` вместе с `ConcurrentHashMap` и `BlockingQueue` .
Когда **ArrayList** `System.arrayCopy()` он создает другой массив и использует `System.arrayCopy()` для копирования всех элементов из одного массива в другой. Это где вставка занимает много времени.
Итератор и ListIterator из Java ArrayList являются **отказоустойчивыми,** это означает, что если ArrayList структурно изменяется в любое время после создания итератора, то любым способом, кроме использования собственных методов удаления или добавления итератора, итератор создает исключение `ConcurrentModificationException` . Таким образом, перед лицом одновременной модификации Iterator дает сбой быстро и чисто, поэтому он называется fail-fast.
`ConcurrentModificationException` не гарантируется и создается только в лучшем виде.
Если вы создаете синхронизированный список , рекомендуется создать его при создании экземпляра базового ArrayList, чтобы предотвратить случайный несинхронизированный доступ к списку.
Приложение может увеличить емкость экземпляра ArrayList перед добавлением большого количества элементов с помощью операции `ensureCapacity()` . Это может уменьшить количество добавочного перераспределения из-за постепенного заполнения ArrayList.
`listIterator()` `size()` , `isEmpty()` , `get()` , `set()` , `iterator()` и `listIterator()` выполняются в постоянное время, **поскольку ArrayList основан на Array,** но добавление или удаление элемента является дорогостоящим по сравнению с `LinkedList` .
Класс ArrayList улучшен в Java 1.5 для поддержки Generics, которые добавили дополнительную безопасность типов в ArrayList. Рекомендуется использовать универсальную версию ArrayList, чтобы гарантировать, что ваш ArrayList содержит только указанный тип элемента, и избегать любых `ClassCastException` .
Поскольку **ArrayList реализует интерфейс List,** он поддерживает порядок вставки элементов, а также допускает дублирование.
Если мы установим для ссылки ArrayList значение `null` в Java, все элементы внутри `ArrayList` получить право на сборку мусора в Java,
при условии, что для этих объектов не существует более строгой ссылки.
Всегда используйте метод `isEmpty()` чтобы проверить, является ли `ArrayList` пустым или нет, вместо использования `size() == 0` check. Бывший гораздо более читабелен, как показано ниже:

```
if(!listOfItems.isEmpty(){
    System.out.println("Starting order processing);
}

if(listOfOrders.size() != 0){
    System.out.println("Order processing started);
}


Read more: http://javarevisited.blogspot.com/2011/05/example-of-arraylist-in-java-tutorial.html#ixzz3wpLtPwop
```

Начиная с Java 5 Tiger, `ArrayList` стал параметризованным, и вы всегда должны использовать универсальную версию этого класса. Это предотвращает классическую ошибку вставки рыбы в список фруктов или вставку кубика в список карточек. При использовании обобщений эти ошибки будут обнаружены компилятором. Следовательно, это также предотвращает `ClassCastException` во время выполнения, потому что компилятор гарантирует, что правильный тип объекта сохраняется и извлекается из Collection. Это также устраняет необходимость в ручном приведении, поскольку компилятор Java автоматически добавляет неявное приведение. Для новичков, понимание дженериков немного сложнее, но стоит учиться, так как в наши дни нет коллекции для использования телом без дженериков.

Когда использовать ArrayList в Java

Теперь вы знаете, что такое ArrayList, различные методы этого класса и как он работает. Пришло время узнать, когда использовать ArrayList. Для программиста на Java не менее важно знать о классах Collection, но не менее важно разработать способность решать, какую коллекцию использовать в конкретном сценарии. В большинстве случаев два фактора определяют ваше решение, производительность и функциональность.

ArrayList – это структура данных на основе индекса, что означает, что он обеспечивает производительность поиска `O(1)` если вы знаете индекс, аналогично добавление элемента в ArrayList также является производительностью `O(1)` в лучшем случае, но если добавление вызывает изменение размера списка, тогда оно будет на уровне `O(n)` потому что столько времени будет потрачено на копирование элементов старого списка в новый ArrayList. Возвращаясь к функциональности, если у вас все в порядке с дублирующимися элементами, используйте только этот класс коллекции. Это не потокобезопасно, поэтому не используйте его в параллельной среде.

Категории

ВЕБ РАЗРАБОТКА	11825
JAVA	7473
ПРОГРАММИРОВАНИЕ	3465
МОБИЛЬНАЯ РАЗРАБОТКА	2983
ПРЕДПРИНИМАТЕЛЬСТВО	2337
DEVOPS	1817
WORDPRESS	1583
ДИЗАЙН	1256
PHP	1191
БАЗЫ ДАННЫХ	947
RUBY	692
УПРАВЛЕНИЕ ПРОЕКТАМИ	232
AI/ML	25
GO	10
NODE.JS	8
ТЕСТИРОВАНИЕ	8
GAMEDEV	7

Последние статьи

СТАТЬИ
Рефакторинг Hudson God Class
Июнь 3, 2019

СТАТЬИ
Альтернативы синтаксиса Java лямбда
Июнь 2, 2019

СТАТЬИ
Morphia и MongoDB: развивающиеся структуры документов
Май 29, 2019

СТАТЬИ
OpenShift Express: развертывание приложения Java EE (с поддержкой AS7)
Май 29, 2019

СТАТЬИ
Интеграция jqGrid, REST, AJAX и Spring MVC
Май 29, 2019