Collections

`Arrays.asList()`

- ***Fixed-size List:*** Because `Arrays.asList()` is ***backed by the original array***. This means you cannot add or remove elements from this List, as these operations would change the size, which is not allowed for an array-backed List. Attempting to do so will result in an `UnsupportedOperationException`.
- ***Backed by the original array:*** Any modifications made to the elements of the returned List will directly reflect in the original array, and vice versa. This is because the List is a view of the array, not a separate copy.
- ***Allows null values:*** Unlike `List.of()`, `Arrays.asList()` permits null values within the array and consequently within the returned List.
- ***Handles primitive arrays differently:*** When used with an array of primitive types (e.g., `int[]`, `char[]`), Arrays.asList() treats the entire primitive array as a single element, creating a List containing only one element, which is the primitive array itself. To create a List of individual primitive values, you need to use wrapper classes (e.g., `Integer[]` instead of `int[]`).

```
Arrays.asList(new int[]{1, 2, 3}); // This gives a List of one element: [{1, 2, 3}]
Arrays.asList(new Integer[]{1, 2, 3}); // This gives a List of elements: [1, 2, 3]
```

`Arrays.asList()` vs `List.of()`

A practical summary of an answer in a Stack Overflow thread:

`Arrays.asList` returns a mutable list while the list returned by `List.of` is structurally immutable:

```
List<Integer> list = Arrays.asList(1, 2, null);
list.set(1, 10); // OK

List<Integer> list = List.of(1, 2, 3);
list.set(1, 10); // Fails with UnsupportedOperationExceptio
```

`Arrays.asList` allows null elements while `List.of` doesn't:

```
List<Integer> list = Arrays.asList(1, 2, null); // OK
List<Integer> list = List.of(1, 2, null); // Fails with Nul
```

`contains` behaves differently with nulls:

```
List<Integer> list = Arrays.asList(1, 2, 3);
list.contains(null); // Returns false

List<Integer> list = List.of(1, 2, 3);
list.contains(null); // Fails with NullPointerException
```

`Arrays.asList` returns a view of the passed array, so the changes to the array will be reflected in the list too. For `List.of` this is not true:

```
Integer[] array = {1,2,3};
List<Integer> list = Arrays.asList(array);
array[1] = 10;
System.out.println(list); // Prints [1, 10, 3]

Integer[] array = {1,2,3};
List<Integer> list = List.of(array);
array[1] = 10;
System.out.println(list); // Prints [1, 2, 3]
```

Share Improve this answer Follow

edited Jun 6, 2024 at 18:12

xuiqzy

315 ● 4 ● 14

answered Oct 5, 2017 at 6:41

ZhekaKozlov

40k ● 20 ● 142 ● 165