Module jdk.incubator.foreign
Package jdk.incubator.foreign

Interface SegmentAllocator

Functional Interface:

This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

```
@FunctionalInterface
public interface SegmentAllocator
```

This interface models a memory allocator. Clients implementing this interface must implement the `allocate(long, long)` method. This interface defines several default methods which can be useful to create segments from several kinds of Java values such as primitives and arrays. This interface can be seen as a thin wrapper around the basic capabilities for creating native segments (e.g. `MemorySegment.allocateNative(long, long, ResourceScope)`); since `SegmentAllocator` is a *functional interface*, clients can easily obtain a native allocator by using either a lambda expression or a method reference.

This interface provides a factory, namely `ofScope(ResourceScope)` which can be used to obtain a *scoped* allocator, that is, an allocator which creates segment bound by a given scope. This can be useful when working inside a *try-with-resources* construct:

```
    try (ResourceScope scope = ResourceScope.newConfinedScope()) {
       SegmentAllocator allocator = SegmentAllocator.ofScope(scope);
       ...
    }
```

In addition, this interface also defines factories for commonly used allocators; for instance `arenaAllocator(ResourceScope)` and `arenaAllocator(long, ResourceScope)` are arena-style native allocators. Finally `ofSegment(MemorySegment)` returns an allocator which wraps a segment (either on-heap or off-heap) and recycles its content upon each new allocation request.

Method Summary

All Methods | Static Methods | Instance Methods | Abstract Methods | Default Methods

Modifier and Type	Method	Description
`default MemorySegment`	`allocate(long bytesSize)`	Allocate a block of memory with given size, with default alignment (1-byte aligned).
`MemorySegment`	`allocate(long bytesSize, long bytesAlignment)`	Allocate a block of memory with given size and alignment constraint.
`default MemorySegment`	`allocate(MemoryLayout layout)`	Allocate a block of memory with given layout.
`default MemorySegment`	`allocate(ValueLayout layout, byte value)`	Allocate a block of memory with given layout and initialize it with given byte value.
`default MemorySegment`	`allocate(ValueLayout layout, char value)`	Allocate a block of memory with given layout and initialize it with given char value.
`default MemorySegment`	`allocate(ValueLayout layout, double value)`	Allocate a block of memory with given layout and initialize it with given double value.
`default MemorySegment`	`allocate(ValueLayout layout, float value)`	Allocate a block of memory with given layout and initialize it with given float value.
`default MemorySegment`	`allocate(ValueLayout layout, int value)`	Allocate a block of memory with given layout and initialize it with given int value.
`default MemorySegment`	`allocate(ValueLayout layout, long value)`	Allocate a block of memory with given layout and initialize it with given long value.
`default MemorySegment`	`allocate(ValueLayout layout, short value)`	Allocate a block of memory with given layout and initialize it with given short value.
`default MemorySegment`	`allocate(ValueLayout layout, Addressable value)`	Allocate a block of memory with given layout and initialize it with given address value (expressed as an `Addressable` instance).
`default MemorySegment`	`allocateArray (MemoryLayout elementLayout, long count)`	Allocate a block of memory corresponding to an array with given element layout and size.
`default MemorySegment`	`allocateArray(ValueLayout elementLayout, byte[] array)`	Allocate a block of memory with given layout and initialize it with given byte array.
`default MemorySegment`	`allocateArray(ValueLayout elementLayout, char[] array)`	Allocate a block of memory with given layout and initialize it with given char array.

default **MemorySegment**	**allocateArray**(**ValueLayout** elementLayout, double[] array)	Allocate a block of memory with given layout and initialize it with given double array.
default **MemorySegment**	**allocateArray**(**ValueLayout** elementLayout, float[] array)	Allocate a block of memory with given layout and initialize it with given float array.
default **MemorySegment**	**allocateArray**(**ValueLayout** elementLayout, int[] array)	Allocate a block of memory with given layout and initialize it with given int array.
default **MemorySegment**	**allocateArray**(**ValueLayout** elementLayout, long[] array)	Allocate a block of memory with given layout and initialize it with given long array.
default **MemorySegment**	**allocateArray**(**ValueLayout** elementLayout, short[] array)	Allocate a block of memory with given layout and initialize it with given short array.
default **MemorySegment**	**allocateArray**(**ValueLayout** elementLayout, **Addressable**[] array)	Allocate a block of memory with given layout and initialize it with given address array.
static **SegmentAllocator**	**arenaAllocator**(long size, **ResourceScope** scope)	Returns a native arena-based allocator which allocates a single memory segment, of given size (using malloc), and then responds to allocation request by returning different slices of that same segment (until no further allocation is possible).
static **SegmentAllocator**	**arenaAllocator**(**ResourceScope** scope)	Returns a native unbounded arena-based allocator.
static **SegmentAllocator**	**ofScope**(**ResourceScope** scope)	Returns a native allocator which responds to allocation requests by allocating new segments bound by the given resource scope, using the `MemorySegment.allocateNative(long, long, ResourceScope)` factory.
static **SegmentAllocator**	**ofSegment**(**MemorySegment** segment)	Returns a segment allocator which responds to allocation requests by recycling a single segment; that is, each new allocation request will return a new slice starting at the segment offset `0` (alignment constraints are ignored by this allocator).

Method Details

allocate

```
default MemorySegment allocate(ValueLayout layout,
                               byte value)
```

Allocate a block of memory with given layout and initialize it with given byte value.

Implementation Requirements:

the default implementation for this method calls `this.allocate(layout)`.

Parameters:

`layout` - the layout of the block of memory to be allocated.

`value` - the value to be set on the newly allocated memory block.

Returns:

a segment for the newly allocated memory block.

Throws:

`IllegalArgumentException` - if `layout.byteSize()` does not conform to the size of a byte value.

allocate

```
default MemorySegment allocate(ValueLayout layout,
                               char value)
```

Allocate a block of memory with given layout and initialize it with given char value.

Implementation Requirements:

the default implementation for this method calls `this.allocate(layout)`.

Parameters:

`layout` - the layout of the block of memory to be allocated.

`value` - the value to be set on the newly allocated memory block.

Returns:

a segment for the newly allocated memory block.

Throws:

`IllegalArgumentException` - if `layout.byteSize()` does not conform to the size of a char value.

allocate

```
default MemorySegment allocate(ValueLayout layout,
                               short value)
```

Allocate a block of memory with given layout and initialize it with given short value.

Implementation Requirements:

the default implementation for this method calls `this.allocate(layout)`.

Parameters:

`layout` - the layout of the block of memory to be allocated.

`value` - the value to be set on the newly allocated memory block.

Returns:

a segment for the newly allocated memory block.

Throws:

`IllegalArgumentException` - if `layout.byteSize()` does not conform to the size of a short value.

allocate

```
default MemorySegment allocate(ValueLayout layout,
                               int value)
```

Allocate a block of memory with given layout and initialize it with given int value.

Implementation Requirements:

the default implementation for this method calls `this.allocate(layout)`.

Parameters:

`layout` - the layout of the block of memory to be allocated.

`value` - the value to be set on the newly allocated memory block.

Returns:

a segment for the newly allocated memory block.

Throws:

`IllegalArgumentException` - if `layout.byteSize()` does not conform to the size of a int value.

allocate

```
default MemorySegment allocate(ValueLayout layout,
                               float value)
```

Allocate a block of memory with given layout and initialize it with given float value.

Implementation Requirements:

the default implementation for this method calls `this.allocate(layout)`.

Parameters:

`layout` - the layout of the block of memory to be allocated.

`value` - the value to be set on the newly allocated memory block.

Returns:

a segment for the newly allocated memory block.

Throws:

`IllegalArgumentException` - if `layout.byteSize()` does not conform to the size of a float value.

allocate

```
default MemorySegment allocate(ValueLayout layout,
                               long value)
```

Allocate a block of memory with given layout and initialize it with given long value.

Implementation Requirements:

the default implementation for this method calls `this.allocate(layout)`.

Parameters:

`layout` - the layout of the block of memory to be allocated.

`value` - the value to be set on the newly allocated memory block.

Returns:

a segment for the newly allocated memory block.

Throws:

`IllegalArgumentException` - if `layout.byteSize()` does not conform to the size of a long value.

allocate

```
default MemorySegment allocate(ValueLayout layout,
                               double value)
```

Allocate a block of memory with given layout and initialize it with given double value.

Implementation Requirements:

the default implementation for this method calls `this.allocate(layout)`.

Parameters:

`layout` - the layout of the block of memory to be allocated.

`value` - the value to be set on the newly allocated memory block.

Returns:

a segment for the newly allocated memory block.

Throws:

`IllegalArgumentException` - if `layout.byteSize()` does not conform to the size of a double value.

allocate

```
default MemorySegment allocate(ValueLayout layout,
                               Addressable value)
```

Allocate a block of memory with given layout and initialize it with given address value (expressed as an `Addressable` instance). The address value might be narrowed according to the platform address size (see `MemoryLayouts.ADDRESS`).

Implementation Requirements:

the default implementation for this method calls `this.allocate(layout)`.

Parameters:

`layout` - the layout of the block of memory to be allocated.

`value` - the value to be set on the newly allocated memory block.

Returns:

a segment for the newly allocated memory block.

Throws:

`IllegalArgumentException` - if `layout.byteSize() != MemoryLayouts.ADDRESS.byteSize()`.

allocateArray

```
default MemorySegment allocateArray(ValueLayout elementLayout,
                                    byte[] array)
```

Allocate a block of memory with given layout and initialize it with given byte array.

Implementation Requirements:

the default implementation for this method calls `this.allocateArray(layout, array.length)`.

Parameters:

`elementLayout` - the element layout of the array to be allocated.

`array` - the array to be copied on the newly allocated memory block.

Returns:

a segment for the newly allocated memory block.

Throws:

`IllegalArgumentException` - if `elementLayout.byteSize()` does not conform to the size of a byte value.

allocateArray

```
default MemorySegment allocateArray(ValueLayout elementLayout,
                                    short[] array)
```

Allocate a block of memory with given layout and initialize it with given short array.

Implementation Requirements:

the default implementation for this method calls `this.allocateArray(layout, array.length)`.

Parameters:

`elementLayout` - the element layout of the array to be allocated.

`array` - the array to be copied on the newly allocated memory block.

Returns:

a segment for the newly allocated memory block.

Throws:

`IllegalArgumentException` - if `elementLayout.byteSize()` does not conform to the size of a short value.

allocateArray

```
default MemorySegment allocateArray(ValueLayout elementLayout,
                                    char[] array)
```

Allocate a block of memory with given layout and initialize it with given char array.

Implementation Requirements:

the default implementation for this method calls `this.allocateArray(layout, array.length)`.

Parameters:

`elementLayout` - the element layout of the array to be allocated.

`array` - the array to be copied on the newly allocated memory block.

Returns:

a segment for the newly allocated memory block.

Throws:

`IllegalArgumentException` - if `elementLayout.byteSize()` does not conform to the size of a char value.

allocateArray

```
default MemorySegment allocateArray(ValueLayout elementLayout,
                                    int[] array)
```

Allocate a block of memory with given layout and initialize it with given int array.

Implementation Requirements:

the default implementation for this method calls `this.allocateArray(layout, array.length)`.

Parameters:

`elementLayout` - the element layout of the array to be allocated.

`array` - the array to be copied on the newly allocated memory block.

Returns:

a segment for the newly allocated memory block.

Throws:

`IllegalArgumentException` - if `elementLayout.byteSize()` does not conform to the size of a int value.

allocateArray

```
default MemorySegment allocateArray(ValueLayout elementLayout,
                                    float[] array)
```

Allocate a block of memory with given layout and initialize it with given float array.

Implementation Requirements:

the default implementation for this method calls `this.allocateArray(layout, array.length)`.

Parameters:

`elementLayout` - the element layout of the array to be allocated.

`array` - the array to be copied on the newly allocated memory block.

Returns:

a segment for the newly allocated memory block.

Throws:

`IllegalArgumentException` - if `elementLayout.byteSize()` does not conform to the size of a float value.

allocateArray

```
default MemorySegment allocateArray(ValueLayout elementLayout,
                                    long[] array)
```

Allocate a block of memory with given layout and initialize it with given long array.

Implementation Requirements:

the default implementation for this method calls `this.allocateArray(layout, array.length)`.

Parameters:

`elementLayout` - the element layout of the array to be allocated.

`array` - the array to be copied on the newly allocated memory block.

Returns:

a segment for the newly allocated memory block.

Throws:

`IllegalArgumentException` - if `elementLayout.byteSize()` does not conform to the size of a long value.

allocateArray

```
default MemorySegment allocateArray(ValueLayout elementLayout,
                                    double[] array)
```

Allocate a block of memory with given layout and initialize it with given double array.

Implementation Requirements:

the default implementation for this method calls `this.allocateArray(layout, array.length)`.

Parameters:

`elementLayout` - the element layout of the array to be allocated.

`array` - the array to be copied on the newly allocated memory block.

Returns:

a segment for the newly allocated memory block.

Throws:

`IllegalArgumentException` - if `elementLayout.byteSize()` does not conform to the size of a double value.

allocateArray

```
default MemorySegment allocateArray(ValueLayout elementLayout,
                                    Addressable[] array)
```

Allocate a block of memory with given layout and initialize it with given address array. The address value of each array element might be narrowed according to the platform address size (see `MemoryLayouts.ADDRESS`).

Implementation Requirements:

the default implementation for this method calls `this.allocateArray(layout, array.length)`.

Parameters:

`elementLayout` - the element layout of the array to be allocated.

`array` - the array to be copied on the newly allocated memory block.

Returns:

a segment for the newly allocated memory block.

Throws:

`IllegalArgumentException` - if `layout.byteSize() != MemoryLayouts.ADDRESS.byteSize()`.

allocate

```
default MemorySegment allocate(MemoryLayout layout)
```

Allocate a block of memory with given layout.

Implementation Requirements:

the default implementation for this method calls `this.allocate(layout.byteSize(), layout.byteAlignment())`.

Parameters:

`layout` - the layout of the block of memory to be allocated.

Returns:

a segment for the newly allocated memory block.

allocateArray

```
default MemorySegment allocateArray(MemoryLayout elementLayout,
                                    long count)
```

Allocate a block of memory corresponding to an array with given element layout and size.

Implementation Requirements:

the default implementation for this method calls `this.allocate(MemoryLayout.sequenceLayout(count, elementLayout))`.

Parameters:

`elementLayout` - the array element layout.

`count` - the array element count.

Returns:

a segment for the newly allocated memory block.

allocate

```
default MemorySegment allocate(long bytesSize)
```

Allocate a block of memory with given size, with default alignment (1-byte aligned).

Implementation Requirements:

the default implementation for this method calls `this.allocate(bytesSize, 1)`.

Parameters:

`bytesSize` - the size (in bytes) of the block of memory to be allocated.

Returns:

a segment for the newly allocated memory block.

allocate

```
MemorySegment allocate(long bytesSize,
                       long bytesAlignment)
```

Allocate a block of memory with given size and alignment constraint.

Parameters:

`bytesSize` - the size (in bytes) of the block of memory to be allocated.

`bytesAlignment` - the alignment (in bytes) of the block of memory to be allocated.

Returns:

a segment for the newly allocated memory block.

arenaAllocator

```
static SegmentAllocator arenaAllocator(long size,
                                       ResourceScope scope)
```

Returns a native arena-based allocator which allocates a single memory segment, of given size (using malloc), and then responds to allocation request by returning different slices of that same segment (until no further allocation is possible). This can be useful when clients want to perform multiple allocation requests while avoiding the cost associated with allocating a new off-heap memory region upon each allocation request.

An allocator associated with a *shared* resource scope is thread-safe and allocation requests may be performed concurrently; conversely, if the arena allocator is associated with a *confined* resource scope, allocation requests can only occur from the thread owning the allocator's resource scope.

The returned allocator might throw an `OutOfMemoryError` if an incoming allocation request exceeds the allocator capacity.

Parameters:

`size` - the size (in bytes) of the allocation arena.

`scope` - the scope associated with the segments returned by this allocator.

Returns:

a new bounded arena-based allocator

Throws:

`IllegalArgumentException` - if `size <= 0`.

`IllegalStateException` - if `scope` has been already closed, or if access occurs from a thread other than the thread owning `scope`.

arenaAllocator

```
static SegmentAllocator arenaAllocator(ResourceScope scope)
```

Returns a native unbounded arena-based allocator.

The returned allocator allocates a memory segment `S` of a certain fixed size (using malloc) and then responds to allocation requests in one of the following ways:

- if the size of the allocation requests is smaller than the size of `S`, and `S` has a *free* slice `S'` which fits that allocation request, return that `S'`.

- if the size of the allocation requests is smaller than the size of `S`, and `S` has no *free* slices which fits that allocation request, allocate a new segment `S'` (using malloc), which has same size as `S` and set `S = S'`; the allocator then tries to respond to the same allocation request again.
- if the size of the allocation requests is bigger than the size of `S`, allocate a new segment `S'` (using malloc), which has a sufficient size to satisfy the allocation request, and return `S'`.

This segment allocator can be useful when clients want to perform multiple allocation requests while avoiding the cost associated with allocating a new off-heap memory region upon each allocation request.

An allocator associated with a *shared* resource scope is thread-safe and allocation requests may be performed concurrently; conversely, if the arena allocator is associated with a *confined* resource scope, allocation requests can only occur from the thread owning the allocator's resource scope.

The returned allocator might throw an `OutOfMemoryError` if an incoming allocation request exceeds the system capacity.

Parameters:

`scope` - the scope associated with the segments returned by this allocator.

Returns:

a new unbounded arena-based allocator

Throws:

`IllegalStateException` - if `scope` has been already closed, or if access occurs from a thread other than the thread owning `scope`.

ofSegment

`static SegmentAllocator ofSegment(MemorySegment segment)`

Returns a segment allocator which responds to allocation requests by recycling a single segment; that is, each new allocation request will return a new slice starting at the segment offset `0` (alignment constraints are ignored by this allocator). This can be useful to limit allocation requests in case a client knows that they have fully processed the contents of the allocated segment before the subsequent allocation request takes place.

While the allocator returned by this method is *thread-safe*, concurrent access on the same recycling allocator might cause a thread to overwrite contents written to the underlying segment by a different thread.

Parameters:

`segment` - the memory segment to be recycled by the returned allocator.

Returns:

an allocator which recycles an existing segment upon each new allocation request.

ofScope

`static SegmentAllocator ofScope(ResourceScope scope)`

Returns a native allocator which responds to allocation requests by allocating new segments bound by the given resource scope, using the `MemorySegment.allocateNative(long, long, ResourceScope)` factory. This code is equivalent (but likely more efficient) to the following:

```
        Resource scope = ...
        SegmentAllocator scoped = (size, align) -> MemorySegment.allocateNative(size, align, scope);
```

Parameters:

`scope` - the resource scope associated with the segments created by the returned allocator.

Returns:

an allocator which allocates new memory segment bound by the provided resource scope.