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; 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 also defines factories for commonly used allocators:

- `nativeAllocator(ResourceScope)` creates an allocator which allocates native segments, backed by a given scope;
- `newNativeArena(ResourceScope)` creates a more efficient arena-style native allocator, where memory is allocated in bigger blocks, which are then sliced accordingly to fit allocation requests;
- `prefixAllocator(MemorySegment)` creates an allocator which wraps a segment (either on-heap or off-heap) and recycles its content upon each new allocation request.

Passing a segment allocator to an API can be especially useful in circumstances where a client wants to communicate *where* the results of a certain operation (performed by the API) should be stored, as a memory segment. For instance, downcall method handles can accept an additional `SegmentAllocator` parameter if the underlying native function is known to return a struct by-value. Effectively, the allocator parameter tells the linker runtime where to store the return value of the native function.

Method Summary

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

Modifier and Type	Method	Description
default `MemorySegment`	**`allocate`**(long bytesSize)	Allocate a memory segment with given size and default alignment constraints (1-byte aligned).
`MemorySegment`	**`allocate`**(long bytesSize, long bytesAlignment)	Allocate a memory segment with given size and alignment constraints.
default `MemorySegment`	**`allocate`**(**`MemoryLayout`** layout)	Allocate a memory segment with given layout.
default `MemorySegment`	**`allocate`**(**`ValueLayout.OfAddress`** layout, **`Addressable`** value)	Allocate a memory segment with given layout and initialize it with given address value (expressed as an `Addressable` instance).
default `MemorySegment`	**`allocate`**(**`ValueLayout.OfByte`** layout, byte value)	Allocate a memory segment with given layout and initialize it with given byte value.
default `MemorySegment`	**`allocate`**(**`ValueLayout.OfChar`** layout, char value)	Allocate a memory segment with given layout and initialize it with given char value.
default `MemorySegment`	**`allocate`**(**`ValueLayout.OfDouble`** layout, double value)	Allocate a memory segment with given layout and initialize it with given double value.
default `MemorySegment`	**`allocate`**(**`ValueLayout.OfFloat`** layout, float value)	Allocate a memory segment with given layout and initialize it with given float value.
default `MemorySegment`	**`allocate`**(**`ValueLayout.OfInt`** layout, int value)	Allocate a memory segment with given layout and initialize it with given int value.
default `MemorySegment`	**`allocate`**(**`ValueLayout.OfLong`** layout, long value)	Allocate a memory segment with given layout and initialize it with given long value.
default `MemorySegment`	**`allocate`**(**`ValueLayout.OfShort`** layout, short value)	Allocate a memory segment with given layout and initialize it with given short value.
default `MemorySegment`	**`allocateArray`**(**`MemoryLayout`** elementLayout, long count)	Allocate a memory segment with given element layout and size.
default `MemorySegment`	**`allocateArray`**(**`ValueLayout.OfByte`** elementLayout, byte[] array)	Allocate a memory segment with given layout and initialize it with given byte array.
default `MemorySegment`	**`allocateArray`**(**`ValueLayout.OfChar`** elementLayout, char[] array)	Allocate a memory segment with given layout and initialize it with given char array.

default **MemorySegment**	**allocateArray**(**ValueLayout.OfDouble** elementLayout, double[] array)	Allocate a memory segment with given layout and initialize it with given double array.
default **MemorySegment**	**allocateArray**(**ValueLayout.OfFloat** elementLayout, float[] array)	Allocate a memory segment with given layout and initialize it with given float array.
default **MemorySegment**	**allocateArray**(**ValueLayout.OfInt** elementLayout, int[] array)	Allocate a memory segment with given layout and initialize it with given int array.
default **MemorySegment**	**allocateArray**(**ValueLayout.OfLong** elementLayout, long[] array)	Allocate a memory segment with given layout and initialize it with given long array.
default **MemorySegment**	**allocateArray**(**ValueLayout.OfShort** elementLayout, short[] array)	Allocate a memory segment with given layout and initialize it with given short array.
default **MemorySegment**	**allocateUtf8String**(**String** str)	Converts a Java string into a UTF-8 encoded, null-terminated C string, storing the result into a memory segment.
static **SegmentAllocator**	**implicitAllocator**()	Returns a native allocator which allocates segments in independent implicit scopes.
static **SegmentAllocator**	**nativeAllocator**(**ResourceScope** scope)	Returns a native allocator, associated with the provided scope.
static **SegmentAllocator**	**newNativeArena**(long arenaSize, long blockSize, **ResourceScope** scope)	Returns a native arena-based allocator, associated with the provided scope, with given arena size and block size.
static **SegmentAllocator**	**newNativeArena**(long arenaSize, **ResourceScope** scope)	Returns a native unbounded arena-based allocator, with block size set to the specified arena size, associated with the provided scope, with given arena size.
static **SegmentAllocator**	**newNativeArena**(**ResourceScope** scope)	Returns a native unbounded arena-based allocator, with predefined block size and maximum arena size, associated with the provided scope.
static **SegmentAllocator**	**prefixAllocator**(**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), hence the name *prefix allocator*.

Method Details

allocateUtf8String

```
default MemorySegment allocateUtf8String(String str)
```

Converts a Java string into a UTF-8 encoded, null-terminated C string, storing the result into a memory segment.

This method always replaces malformed-input and unmappable-character sequences with this charset's default replacement byte array. The `CharsetEncoder` class should be used when more control over the encoding process is required.

Implementation Requirements:

the default implementation for this method copies the contents of the provided Java string into a new memory segment obtained by calling `this.allocate(str.length() + 1)`.

Parameters:

`str` - the Java string to be converted into a C string.

Returns:

a new native memory segment containing the converted C string.

allocate

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

Allocate a memory segment 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.

allocate

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

Allocate a memory segment 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.

allocate

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

Allocate a memory segment 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.

allocate

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

Allocate a memory segment 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.

allocate

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

Allocate a memory segment 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.

allocate

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

Allocate a memory segment 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.

allocate

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

Allocate a memory segment 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.

allocate

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

Allocate a memory segment 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 `ValueLayout.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.

allocateArray

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

Allocate a memory segment 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.

allocateArray

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

Allocate a memory segment 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.

allocateArray

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

Allocate a memory segment 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.

allocateArray

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

Allocate a memory segment 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.

allocateArray

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

Allocate a memory segment 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.

allocateArray

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

Allocate a memory segment 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.

allocateArray

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

Allocate a memory segment 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.

allocate

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

Allocate a memory segment 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 memory segment 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 memory segment with given size and default alignment constraints (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 memory segment with given size and alignment constraints.

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.

newNativeArena

```
static SegmentAllocator newNativeArena(ResourceScope scope)
```

Returns a native unbounded arena-based allocator, with predefined block size and maximum arena size, associated with the provided scope. Equivalent to the following code:

```
SegmentAllocator.newNativeArena(Long.MAX_VALUE, predefinedBlockSize, scope);
```

Parameters:

`scope` - the scope associated with the segments returned by the arena-based 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`.

newNativeArena

```
static SegmentAllocator newNativeArena(long arenaSize,
                                       ResourceScope scope)
```

Returns a native unbounded arena-based allocator, with block size set to the specified arena size, associated with the provided scope, with given arena size. Equivalent to the following code:

```
SegmentAllocator.newNativeArena(arenaSize, arenaSize, scope);
```

Parameters:

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

`scope` - the scope associated with the segments returned by the arena-based allocator.

Returns:

a new unbounded arena-based allocator

Throws:

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

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

newNativeArena

```
static SegmentAllocator newNativeArena(long arenaSize,
                                       long blockSize,
                                       ResourceScope scope)
```

Returns a native arena-based allocator, associated with the provided scope, with given arena size and block size.

The returned allocator allocates a memory segment `S` of the specified block size 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'`, 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'`, 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.

The returned allocator might throw an `OutOfMemoryError` if the total memory allocated with this allocator exceeds the arena size, or the system capacity. Furthermore, the returned allocator is not thread safe. Concurrent allocation needs to be guarded with synchronization primitives.

Parameters:

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

`blockSize` - the block size associated with the arena-based allocator.

`scope` - the scope associated with the segments returned by the arena-based allocator.

Returns:

a new unbounded arena-based allocator

Throws:

`IllegalArgumentException` - if `blockSize <= 0`, if `arenaSize <= 0` or if `arenaSize < blockSize`.

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

prefixAllocator

```
static SegmentAllocator prefixAllocator(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), hence the name *prefix allocator*. Equivalent to (but likely more efficient than) the following code:

```
MemorySegment segment = ...
SegmentAllocator prefixAllocator = (size, align) -> segment.asSlice(0, size);
```

This allocator 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.

nativeAllocator

```
static SegmentAllocator nativeAllocator(ResourceScope scope)
```

Returns a native allocator, associated with the provided scope. Equivalent to (but likely more efficient than) the following code:

```
ResourceScope scope = ...
SegmentAllocator nativeAllocator = (size, align) -> MemorySegment.allocateNative(size, align, scope);
```

Parameters:

`scope` - the scope associated with the returned allocator.

Returns:

a native allocator, associated with the provided scope.

implicitAllocator

```
static SegmentAllocator implicitAllocator()
```

Returns a native allocator which allocates segments in independent implicit scopes. Equivalent to (but likely more efficient than) the following code:

```
SegmentAllocator implicitAllocator = (size, align) -> MemorySegment.allocateNative(size, align, ResourceS
```

Returns:

a native allocator which allocates segments in independent implicit scopes.