Module java.base
Package java.lang.foreign

Interface MemorySession

All Superinterfaces:

`AutoCloseable`, `SegmentAllocator`PREVIEW

```
public sealed interface MemorySession
extends AutoCloseable, SegmentAllocatorPREVIEW
```

`MemorySession` is a preview API of the Java platform.
Programs can only use `MemorySession` when preview features are enabled.
Preview features may be removed in a future release, or upgraded to permanent features of the Java platform.

A memory session manages the lifecycle of one or more resources. Resources (e.g. `MemorySegment`PREVIEW) associated with a memory session can only be accessed while the memory session is alive, and by the thread associated with the memory session (if any).

Memory sessions can be closed. When a memory session is closed, it is no longer alive, and subsequent operations on resources associated with that session (e.g. attempting to access a `MemorySegment`PREVIEW instance) will fail with `IllegalStateException`.

A memory session is associated with one or more close actions. Close actions can be used to specify the cleanup code that must run when a given resource (or set of resources) is no longer in use. When a memory session is closed, the close actions associated with that session are executed (in unspecified order). For instance, closing the memory session associated with one or more native memory segmentsPREVIEW results in releasing the off-heap memory associated with said segments.

The global session is a memory session that cannot be closed. As a result, resources associated with the global session are never released. Examples of resources associated with the global memory session are heap segmentsPREVIEW.

Thread confinement

Memory sessions can be divided into two categories: *thread-confined* memory sessions, and *shared* memory sessions.

Confined memory sessions, support strong thread-confinement guarantees. Upon creation, they are assigned an owner thread, typically the thread which initiated the creation operation. After creating a confined memory session, only the owner thread will be allowed to directly manipulate the resources associated with this memory session. Any attempt to perform resource access from a thread other than the owner thread will fail with `WrongThreadException`.

Shared memory sessions, on the other hand, have no owner thread; as such, resources associated with shared memory sessions can be accessed by multiple threads. This might be useful when multiple threads need to access the same resource concurrently (e.g. in the case of parallel processing).

Closeable memory sessions

When a session is associated with off-heap resources, it is often desirable for said resources to be released in a timely fashion, rather than waiting for the session to be deemed unreachable by the garbage collector. In this scenario, a client might consider using a *closeable* memory session. Closeable memory sessions are memory sessions that can be closed deterministically, as demonstrated in the following example:

```
try (MemorySession session = MemorySession.openConfined()) {
   MemorySegment segment1 = MemorySegment.allocateNative(100);
   MemorySegment segment1 = MemorySegment.allocateNative(200);
   ...
} // all memory released here
```

The above code creates a confined, closeable session. Then it allocates two segments associated with that session. When the session is closed (above, this is done implicitly, using the *try-with-resources construct*), all memory allocated within the session will be released

Closeable memory sessions, while powerful, must be used with caution. Closeable memory sessions must be closed when no longer in use, either explicitly (by calling the `close()` method), or implicitly (by wrapping the use of a closeable memory session in a *try-with-resources construct*). A failure to do so might result in memory leaks. To mitigate this problem, closeable memory sessions can be associated with a `Cleaner` instance, so that they are also closed automatically, once the session instance becomes unreachable. This can be useful to allow for predictable, deterministic resource deallocation, while still preventing accidental native memory leaks. In case a client closes a memory session managed by a cleaner, no further action will be taken when the session becomes unreachable; that is, close actions associated with a memory session, whether managed or not, are called *exactly once*.

Non-closeable views

There are situations in which it might not be desirable for a memory session to be reachable from one or more resources associated with it. For instance, an API might create a private memory session, and allocate a memory segment, and then expose one or more slices of this segment to its clients. Since the API's memory session would be reachable from the slices (using the `MemorySegment.session()`PREVIEW accessor), it might be possible for clients to compromise the API (e.g. by closing the session prematurely). To avoid leaking private memory sessions to untrusted clients, an API can instead return segments based on a non-closeable view of the session it created, as follows:

```
MemorySession session = MemorySession.openConfined();
MemorySession nonCloseableSession = session.asNonCloseable();
MemorySegment segment = MemorySegment.allocateNative(100, nonCloseableSession);
segment.session().close(); // throws
session.close(); //ok
```

In other words, only the owner of the original `session` object can close the session. External clients can only access the non-closeable session, and have no access to the underlying API session.

Implementation Requirements:

Implementations of this interface are thread-safe.

Since:

19

See Also:

`MemorySegment`PREVIEW, `SymbolLookup`PREVIEW, `Linker`PREVIEW, `VaList`PREVIEW

Method Summary

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

Modifier and Type	Method	Description
`void`	`addCloseAction(Runnable runnable)`	Adds a custom cleanup action which will be executed when the memory session is closed.
`default MemorySegment`PREVIEW	`allocate(long bytesSize, long bytesAlignment)`	Allocates a native segment, using this session.
`MemorySession`PREVIEW	`asNonCloseable()`	Returns a non-closeable view of this memory session.
`void`	`close()`	Closes this memory session.
`boolean`	`equals(Object that)`	Compares the specified object with this memory session for equality.
`static MemorySession`PREVIEW	`global()`	Returns the global memory session.
`int`	`hashCode()`	Returns the hash code value for this memory session.
`boolean`	`isAlive()`	Returns `true`, if this memory session is alive.
`boolean`	`isCloseable()`	Returns `true`, if this session is a closeable memory session.
`static MemorySession`PREVIEW	`openConfined()`	Creates a closeable confined memory session.
`static MemorySession`PREVIEW	`openConfined(Cleaner cleaner)`	Creates a closeable confined memory session, managed by the provided cleaner instance.
`static MemorySession`PREVIEW	`openImplicit()`	Creates a non-closeable shared memory session, managed by a private `Cleaner` instance.
`static MemorySession`PREVIEW	`openShared()`	Creates a closeable shared memory session.
`static MemorySession`PREVIEW	`openShared(Cleaner cleaner)`	Creates a closeable shared memory session, managed by the provided cleaner instance.
`Thread`	`ownerThread()`	Returns the owner thread associated with this memory session, or `null` if this session is shared across multiple threads.
`void`	`whileAlive(Runnable action)`	Runs a critical action while this memory session is kept alive.

Methods declared in interface java.lang.foreign.SegmentAllocatorPREVIEW

`allocate, allocate, allocate, allocate, allocate, allocate, allocate, allocate, allocate, allocate, allocateArray, allocateArray, allocateArray, allocateArray, allocateArray, allocateArray, allocateArray, allocateArray, allocateUtf8String`

Method Details

isAlive

```
boolean isAlive()
```

Returns `true`, if this memory session is alive.

Returns:

`true`, if this memory session is alive

isCloseable

```
boolean isCloseable()
```

Returns `true`, if this session is a closeable memory session..

Returns:

`true`, if this session is a closeable memory session

ownerThread

```
Thread ownerThread()
```

Returns the owner thread associated with this memory session, or `null` if this session is shared across multiple threads.

Returns:

the owner thread associated with this memory session, or `null` if this session is shared across multiple threads

whileAlive

```
void whileAlive(Runnable action)
```

Runs a critical action while this memory session is kept alive.

Parameters:

`action` - the action to be run.

addCloseAction

```
void addCloseAction(Runnable runnable)
```

Adds a custom cleanup action which will be executed when the memory session is closed. The order in which custom cleanup actions are invoked once the memory session is closed is unspecified.

API Note:

The provided action should not keep a strong reference to this memory session, so that implicitly closed sessions can be handled correctly by a `Cleaner` instance.

Parameters:

`runnable` - the custom cleanup action to be associated with this memory session.

Throws:

`IllegalStateException` - if this memory session is not alive.

`WrongThreadException` - if this method is called from a thread other than the thread owning this memory session.

close

```
void close()
```

Closes this memory session. If this operation completes without exceptions, this session will be marked as *not alive*, the close actions associated with this session will be executed, and all the resources associated with this session will be released.

Specified by:

`close` in interface `AutoCloseable`

API Note:

This operation is not idempotent; that is, closing an already closed memory session *always* results in an exception being thrown. This reflects a deliberate design choice: memory session state transitions should be manifest in the client code; a failure in any of these transitions reveals a bug in the underlying application logic.

Throws:

`IllegalStateException` - if this memory session is not alive.

`IllegalStateException` - if this session is kept alive by another client.

`WrongThreadException` - if this method is called from a thread other than the thread owning this memory session.

`UnsupportedOperationException` - if this memory session is not closeable.

See Also:

`isAlive()`

asNonCloseable

```
MemorySessionPREVIEW asNonCloseable()
```

Returns a non-closeable view of this memory session. If this session is non-closeable, this session is returned. Otherwise, this method returns a non-closeable view of this memory session.

API Note:

a non-closeable view of a memory session `S` keeps `S` reachable. As such, `S` cannot be closed implicitly (e.g. by a `Cleaner`) as long as one or more non-closeable views of `S` are reachable.

Returns:

a non-closeable view of this memory session.

equals

```
boolean equals(Object that)
```

Compares the specified object with this memory session for equality. Returns `true` if and only if the specified object is also a memory session, and it refers to the same memory session as this memory session. A non-closeable view `V` of a memory session `S` is considered equal to `S`.

Overrides:

`equals` in class `Object`

Parameters:

`that` - the object to be compared for equality with this memory session.

Returns:

`true` if the specified object is equal to this memory session.

See Also:

`Object.hashCode()`, `HashMap`

hashCode

```
int hashCode()
```

Returns the hash code value for this memory session.

Overrides:

`hashCode` in class `Object`

Returns:

the hash code value for this memory session

See Also:

`Object.equals(java.lang.Object)`,
`System.identityHashCode(java.lang.Object)`

allocate

```
default MemorySegmentPREVIEW allocate(long bytesSize,
                                      long bytesAlignment)
```

Allocates a native segment, using this session. Equivalent to the following code:

```
MemorySegment.allocateNative(size, align, this);
```

Specified by:

`allocate` in interface `SegmentAllocatorPREVIEW`

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 new native segment, associated with this session.

Throws:

`IllegalStateException` - if this memory session is not alive.

`WrongThreadException` - if this method is called from a thread other than the thread owning this memory session.

openConfined

`static MemorySessionPREVIEW openConfined()`

Creates a closeable confined memory session.

Returns:
a new closeable confined memory session.

openConfined

`static MemorySessionPREVIEW openConfined(Cleaner cleaner)`

Creates a closeable confined memory session, managed by the provided cleaner instance.

Parameters:
`cleaner` - the cleaner to be associated with the returned memory session.

Returns:
a new closeable confined memory session, managed by `cleaner`.

openShared

`static MemorySessionPREVIEW openShared()`

Creates a closeable shared memory session.

Returns:
a new closeable shared memory session.

openShared

`static MemorySessionPREVIEW openShared(Cleaner cleaner)`

Creates a closeable shared memory session, managed by the provided cleaner instance.

Parameters:
`cleaner` - the cleaner to be associated with the returned memory session.

Returns:
a new closeable shared memory session, managed by `cleaner`.

openImplicit

`static MemorySessionPREVIEW openImplicit()`

Creates a non-closeable shared memory session, managed by a private `Cleaner` instance. Equivalent to (but likely more efficient than) the following code:

```
openShared(Cleaner.create()).asNonCloseable();
```

Returns:
a non-closeable shared memory session, managed by a private `Cleaner` instance.

global

`static MemorySessionPREVIEW global()`

Returns the global memory session.

Returns:
the global memory session.