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

Interface SymbolLookup

All Known Subinterfaces:

CLinker

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 SymbolLookup
```

A symbol lookup. Exposes a lookup operation for searching symbol addresses by name, see `lookup(String)`. A symbol lookup can be used to look up a symbol in a loaded library. Clients can obtain a loader lookup, which can be used to search symbols in libraries loaded by the current classloader (e.g. using `System.load(String)`, or `System.loadLibrary(String)`). Alternatively, clients can search symbols in the standard C library using a `CLinker`, which conveniently implements this interface.

Unless otherwise specified, passing a `null` argument, or an array argument containing one or more `null` elements to a method in this class causes a `NullPointerException` to be thrown.

Method Summary

All Methods | Static Methods | Instance Methods | Abstract Methods

Modifier and Type	Method	Description
`static SymbolLookup`	`loaderLookup()`	Obtains a symbol lookup suitable to find symbols in native libraries associated with the caller's classloader (that is, libraries loaded using `System.loadLibrary(java.lang.String)` or `System.load(java.lang.String)`).
`Optional<NativeSymbol>`	`lookup(String name)`	Looks up a symbol with given name in this lookup.

Method Details

lookup

`Optional<NativeSymbol> lookup(String name)`

Looks up a symbol with given name in this lookup.

Parameters:

`name` - the symbol name.

Returns:

the lookup symbol (if any).

loaderLookup

`static SymbolLookup loaderLookup()`

Obtains a symbol lookup suitable to find symbols in native libraries associated with the caller's classloader (that is, libraries loaded using `System.loadLibrary(java.lang.String)` or `System.load(java.lang.String)`). The returned lookup returns native symbols backed by a non-closeable, shared scope which keeps the caller's classloader reachable.

This method is *restricted*. Restricted methods are unsafe, and, if used incorrectly, their use might crash the JVM or, worse, silently result in memory corruption. Thus, clients should refrain from depending on restricted methods, and use safe and supported functionalities, where possible.

Returns:

a symbol lookup suitable to find symbols in libraries loaded by the caller's classloader.

Throws:

`IllegalCallerException` - if access to this method occurs from a module `M` and the command line option `--enable-native-access` is either absent, or does not mention the module name `M`, or `ALL-UNNAMED` in case `M` is an unnamed module.
