Module java.base
Package java.lang.invoke

Class MethodType

java.lang.Object
    java.lang.invoke.MethodType

All Implemented Interfaces:

Serializable, Constable, TypeDescriptor, TypeDescriptor.OfMethod<Class<?>, MethodType>

```
public final class MethodType
extends Object
implements Constable, TypeDescriptor.OfMethod<Class<?>,MethodType>, Serializable
```

A method type represents the arguments and return type accepted and returned by a method handle, or the arguments and return type passed and expected by a method handle caller. Method types must be properly matched between a method handle and all its callers, and the JVM's operations enforce this matching at, specifically during calls to `MethodHandle.invokeExact` and `MethodHandle.invoke`, and during execution of `invokedynamic` instructions.

The structure is a return type accompanied by any number of parameter types. The types (primitive, `void`, and reference) are represented by `Class` objects. (For ease of exposition, we treat `void` as if it were a type. In fact, it denotes the absence of a return type.)

All instances of `MethodType` are immutable. Two instances are completely interchangeable if they compare equal. Equality depends on pairwise correspondence of the return and parameter types and on nothing else.

This type can be created only by factory methods. All factory methods may cache values, though caching is not guaranteed. Some factory methods are static, while others are virtual methods which modify precursor method types, e.g., by changing a selected parameter.

Factory methods which operate on groups of parameter types are systematically presented in two versions, so that both Java arrays and Java lists can be used to work with groups of parameter types. The query methods `parameterArray` and `parameterList` also provide a choice between arrays and lists.

`MethodType` objects are sometimes derived from bytecode instructions such as `invokedynamic`, specifically from the type descriptor strings associated with the instructions in a class file's constant pool.

Like classes and strings, method types can also be represented directly in a class file's constant pool as constants. A method type may be loaded by an `ldc` instruction which refers to a suitable `CONSTANT_MethodType` constant pool entry. The entry refers to a `CONSTANT_Utf8` spelling for the descriptor string. (For full details on method type constants, see sections 4.4.8 and 5.4.3.5 of the Java Virtual Machine Specification.)

When the JVM materializes a `MethodType` from a descriptor string, all classes named in the descriptor must be accessible, and will be loaded. (But the classes need not be initialized, as is the case with a `CONSTANT_Class`.) This loading may occur at any time before the `MethodType` object is first derived.

Nominal Descriptors

A `MethodType` can be described in nominal form if and only if all of the parameter types and return type can be described with a `nominal descriptor` represented by `ClassDesc`. If a method type can be described nominally, then:

- The method type has a `nominal descriptor` returned by `MethodType::describeConstable`.
- The descriptor string returned by `MethodType::descriptorString` or `MethodType::toMethodDescriptorString` for the method type is a method descriptor (JVMS 4.3.3).

If any of the parameter types or return type cannot be described nominally, i.e. `Class::describeConstable` returns an empty optional for that type, then the method type cannot be described nominally:

- The method type has no `nominal descriptor` and `MethodType::describeConstable` returns an empty optional.
- The descriptor string returned by `MethodType::descriptorString` or `MethodType::toMethodDescriptorString` for the method type is not a type descriptor.

Since:

1.7

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces declared in interface java.lang.invoke.TypeDescriptor

`TypeDescriptor.OfField<F extends TypeDescriptor.OfField<F>>,`
`TypeDescriptor.OfMethod<F extends TypeDescriptor.OfField<F>,M extends TypeDescriptor.OfMethod<F,M>>`

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
`MethodType`	`appendParameterTypes(Class<? >... ptypesToInsert)`	Finds or creates a method type with additional parameter types.
`MethodType`	`appendParameterTypes(List<Class<? >> ptypesToInsert)`	Finds or creates a method type with additional parameter types.
`MethodType`	`changeParameterType(int num, Class<? > nptype)`	Finds or creates a method type with a single different parameter type.

MethodType	changeReturnType(Class<? > nrtype)	Finds or creates a method type with a different return type.
Optional<MethodTypeDes	describeConstable()	Returns a nominal descriptor for this instance, if one can be constructed, or an empty Optional if one cannot be.
String	descriptorString()	Returns a descriptor string for this method type.
MethodType	dropParameterTypes (int start, int end)	Finds or creates a method type with some parameter types omitted.
boolean	equals(Object x)	Compares the specified object with this type for equality.
MethodType	erase()	Erases all reference types to Object.
static MethodType	fromMethodDescriptorString (String descriptor, ClassLoader loader)	Finds or creates an instance of a method type, given the spelling of its bytecode descriptor.
MethodType	generic()	Converts all types, both reference and primitive, to Object.
static MethodType	genericMethodType (int objectArgCount)	Finds or creates a method type whose components are all Object.
static MethodType	genericMethodType (int objectArgCount, boolean finalArray)	Finds or creates a method type whose components are Object with an optional trailing Object[] array.
int	hashCode()	Returns the hash code value for this method type.
boolean	hasPrimitives()	Reports if this type contains a primitive argument or return value.
boolean	hasWrappers()	Reports if this type contains a wrapper argument or return value.
MethodType	insertParameterTypes (int num, Class<?	Finds or creates a method type with additional

	>... ptypesToInsert)	parameter types.
MethodType	insertParameterTypes (int num, List<Class<? >> ptypesToInsert)	Finds or creates a method type with additional parameter types.
Class<?>	lastParameterType()	Returns the last parameter type of this method type.
static MethodType	methodType(Class<? > rtype)	Finds or creates a method type with the given components.
static MethodType	methodType(Class<? > rtype, Class<?> ptype0)	Finds or creates a method type with the given components.
static MethodType	methodType(Class<? > rtype, Class<?> [] ptypes)	Finds or creates an instance of the given method type.
static MethodType	methodType(Class<? > rtype, Class<?> ptype0, Class<?>... ptypes)	Finds or creates a method type with the given components.
static MethodType	methodType(Class<? > rtype, MethodType ptypes)	Finds or creates a method type with the given components.
static MethodType	methodType(Class<? > rtype, List<Class<? >> ptypes)	Finds or creates a method type with the given components.
Class<?>[]	parameterArray()	Presents the parameter types as an array (a convenience method).
int	parameterCount()	Returns the number of parameter types in this method type.
List<Class<?>>	parameterList()	Presents the parameter types as a list (a convenience method).
Class<?>	parameterType(int num)	Returns the parameter type at the specified index, within this method type.
Class<?>	returnType()	Returns the return type of this method type.
String	toMethodDescriptorString()	Returns a descriptor string for the method type.
String	toString()	Returns a string

		representation of the method type, of the form "(PT0,PT1...)RT".
MethodType	unwrap()	Converts all wrapper types to their corresponding primitive types.
MethodType	wrap()	Converts all primitive types to their corresponding wrapper types.

Methods declared in class java.lang.Object

clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Method Details

methodType

```
public static MethodType methodType(Class<?> rtype,
                                    Class<?>[] ptypes)
```

Finds or creates an instance of the given method type.

Parameters:

rtype - the return type

ptypes - the parameter types

Returns:

a method type with the given components

Throws:

NullPointerException - if rtype or ptypes or any element of ptypes is null

IllegalArgumentException - if any element of ptypes is void.class

methodType

```
public static MethodType methodType(Class<?> rtype,
                                    List<Class<?>> ptypes)
```

Finds or creates a method type with the given components. Convenience method for methodType.

Parameters:

rtype - the return type

ptypes - the parameter types

Returns:

a method type with the given components

Throws:

`NullPointerException` - if `rtype` or `ptypes` or any element of `ptypes` is null

`IllegalArgumentException` - if any element of `ptypes` is `void.class`

methodType

```
public static MethodType methodType(Class<?> rtype,
                                    Class<?> ptype0,
                                    Class<?>... ptypes)
```

Finds or creates a method type with the given components. Convenience method for `methodType`. The leading parameter type is prepended to the remaining array.

Parameters:

`rtype` - the return type

`ptype0` - the first parameter type

`ptypes` - the remaining parameter types

Returns:

a method type with the given components

Throws:

`NullPointerException` - if `rtype` or `ptype0` or `ptypes` or any element of `ptypes` is null

`IllegalArgumentException` - if `ptype0` or `ptypes` or any element of `ptypes` is `void.class`

methodType

```
public static MethodType methodType(Class<?> rtype)
```

Finds or creates a method type with the given components. Convenience method for `methodType`. The resulting method has no parameter types.

Parameters:

`rtype` - the return type

Returns:

a method type with the given return value

Throws:

`NullPointerException` - if `rtype` is null

methodType

```
public static MethodType methodType(Class<?> rtype,
                                    Class<?> ptype0)
```

Finds or creates a method type with the given components. Convenience method for `methodType`. The resulting method has the single given parameter type.

Parameters:

`rtype` - the return type

`ptype0` - the parameter type

Returns:

a method type with the given return value and parameter type

Throws:

`NullPointerException` - if `rtype` or `ptype0` is null

`IllegalArgumentException` - if `ptype0` is `void.class`

methodType

```
public static MethodType methodType(Class<?> rtype,
                                    MethodType ptypes)
```

Finds or creates a method type with the given components. Convenience method for `methodType`. The resulting method has the same parameter types as `ptypes`, and the specified return type.

Parameters:

`rtype` - the return type

`ptypes` - the method type which supplies the parameter types

Returns:

a method type with the given components

Throws:

`NullPointerException` - if `rtype` or `ptypes` is null

genericMethodType

```
public static MethodType genericMethodType(int objectArgCount,
                                           boolean finalArray)
```

Finds or creates a method type whose components are `Object` with an optional trailing `Object[]` array. Convenience method for `methodType`. All parameters and the return type will be `Object`, except the final array parameter if any, which will be `Object[]`.

Parameters:

`objectArgCount` - number of parameters (excluding the final array parameter if any)

`finalArray` - whether there will be a trailing array parameter, of type `Object[]`

Returns:

a generally applicable method type, for all calls of the given fixed argument count and a collected array of further arguments

Throws:

`IllegalArgumentException` - if `objectArgCount` is negative or greater than 255 (or 254, if `finalArray` is true)

See Also:

genericMethodType(int)

genericMethodType

public static MethodType genericMethodType(int objectArgCount)

Finds or creates a method type whose components are all Object. Convenience method for methodType. All parameters and the return type will be Object.

Parameters:

objectArgCount - number of parameters

Returns:

a generally applicable method type, for all calls of the given argument count

Throws:

IllegalArgumentException - if objectArgCount is negative or greater than 255

See Also:

genericMethodType(int, boolean)

changeParameterType

public MethodType changeParameterType(int num,
Class<?> nptype)

Finds or creates a method type with a single different parameter type. Convenience method for methodType.

Specified by:

changeParameterType in interface TypeDescriptor.OfMethod<Class<?>,MethodType>

Parameters:

num - the index (zero-based) of the parameter type to change

nptype - a new parameter type to replace the old one with

Returns:

the same type, except with the selected parameter changed

Throws:

IndexOutOfBoundsException - if num is not a valid index into parameterArray()

IllegalArgumentException - if nptype is void.class

NullPointerException - if nptype is null

insertParameterTypes

public MethodType insertParameterTypes(int num,
Class<?>... ptypesToInsert)

Finds or creates a method type with additional parameter types. Convenience method for methodType.

Specified by:

`insertParameterTypes` in interface `TypeDescriptor.OfMethod<Class<?>,MethodType>`

Parameters:

`num` - the position (zero-based) of the inserted parameter type(s)

`ptypesToInsert` - zero or more new parameter types to insert into the parameter list

Returns:

the same type, except with the selected parameter(s) inserted

Throws:

`IndexOutOfBoundsException` - if `num` is negative or greater than `parameterCount()`

`IllegalArgumentException` - if any element of `ptypesToInsert` is `void.class` or if the resulting method type would have more than 255 parameter slots

`NullPointerException` - if `ptypesToInsert` or any of its elements is null

appendParameterTypes

```
public MethodType appendParameterTypes(Class<?>... ptypesToInsert)
```

Finds or creates a method type with additional parameter types. Convenience method for `methodType`.

Parameters:

`ptypesToInsert` - zero or more new parameter types to insert after the end of the parameter list

Returns:

the same type, except with the selected parameter(s) appended

Throws:

`IllegalArgumentException` - if any element of `ptypesToInsert` is `void.class` or if the resulting method type would have more than 255 parameter slots

`NullPointerException` - if `ptypesToInsert` or any of its elements is null

insertParameterTypes

```
public MethodType insertParameterTypes(int num,
                                       List<Class<?>> ptypesToInsert)
```

Finds or creates a method type with additional parameter types. Convenience method for `methodType`.

Parameters:

`num` - the position (zero-based) of the inserted parameter type(s)

`ptypesToInsert` - zero or more new parameter types to insert into the parameter list

Returns:

the same type, except with the selected parameter(s) inserted

Throws:

IndexOutOfBoundsException - if num is negative or greater than parameterCount()

IllegalArgumentException - if any element of ptypesToInsert is void.class or if the resulting method type would have more than 255 parameter slots

NullPointerException - if ptypesToInsert or any of its elements is null

appendParameterTypes

```
public MethodType appendParameterTypes(List<Class<?>> ptypesToInsert)
```

Finds or creates a method type with additional parameter types. Convenience method for methodType.

Parameters:

ptypesToInsert - zero or more new parameter types to insert after the end of the parameter list

Returns:

the same type, except with the selected parameter(s) appended

Throws:

IllegalArgumentException - if any element of ptypesToInsert is void.class or if the resulting method type would have more than 255 parameter slots

NullPointerException - if ptypesToInsert or any of its elements is null

dropParameterTypes

```
public MethodType dropParameterTypes(int start,
                                     int end)
```

Finds or creates a method type with some parameter types omitted. Convenience method for methodType.

Specified by:

dropParameterTypes in interface TypeDescriptor.OfMethod<Class<?>,MethodType>

Parameters:

start - the index (zero-based) of the first parameter type to remove

end - the index (greater than start) of the first parameter type after not to remove

Returns:

the same type, except with the selected parameter(s) removed

Throws:

IndexOutOfBoundsException - if start is negative or greater than parameterCount() or if end is negative or greater than parameterCount() or if start is greater than end

changeReturnType

```
public MethodType changeReturnType(Class<?> nrtype)
```

Finds or creates a method type with a different return type. Convenience method for `methodType`.

Specified by:

`changeReturnType` in interface `TypeDescriptor.OfMethod<Class<?>,MethodType>`

Parameters:

`nrtype` - a return parameter type to replace the old one with

Returns:

the same type, except with the return type change

Throws:

`NullPointerException` - if `nrtype` is null

hasPrimitives

```
public boolean hasPrimitives()
```

Reports if this type contains a primitive argument or return value. The return type `void` counts as a primitive.

Returns:

true if any of the types are primitives

hasWrappers

```
public boolean hasWrappers()
```

Reports if this type contains a wrapper argument or return value. Wrappers are types which box primitive values, such as `Integer`. The reference type `java.lang.Void` counts as a wrapper, if it occurs as a return type.

Returns:

true if any of the types are wrappers

erase

```
public MethodType erase()
```

Erases all reference types to `Object`. Convenience method for `methodType`. All primitive types (including `void`) will remain unchanged.

Returns:

a version of the original type with all reference types replaced

generic

```
public MethodType generic()
```

Converts all types, both reference and primitive, to `Object`. Convenience method for `genericMethodType`. The expression `type.wrap().erase()` produces the same value as `type.generic()`.

Returns:

a version of the original type with all types replaced

wrap

```
public MethodType wrap()
```

Converts all primitive types to their corresponding wrapper types. Convenience method for `methodType`. All reference types (including wrapper types) will remain unchanged. A `void` return type is changed to the type `java.lang.Void`. The expression `type.wrap().erase()` produces the same value as `type.generic()`.

Returns:

a version of the original type with all primitive types replaced

unwrap

```
public MethodType unwrap()
```

Converts all wrapper types to their corresponding primitive types. Convenience method for `methodType`. All primitive types (including `void`) will remain unchanged. A return type of `java.lang.Void` is changed to `void`.

Returns:

a version of the original type with all wrapper types replaced

parameterType

```
public Class<?> parameterType(int num)
```

Returns the parameter type at the specified index, within this method type.

Specified by:

`parameterType` in interface `TypeDescriptor.OfMethod<Class<?>,MethodType>`

Parameters:

`num` - the index (zero-based) of the desired parameter type

Returns:

the selected parameter type

Throws:

`IndexOutOfBoundsException` - if `num` is not a valid index into `parameterArray()`

parameterCount

```
public int parameterCount()
```

Returns the number of parameter types in this method type.

Specified by:

`parameterCount` in interface `TypeDescriptor.OfMethod<Class<?>,MethodType>`

Returns:

the number of parameter types

returnType

```
public Class<?> returnType()
```

Returns the return type of this method type.

Specified by:

`returnType` in interface `TypeDescriptor.OfMethod<Class<?>,MethodType>`

Returns:

the return type

parameterList

```
public List<Class<?>> parameterList()
```

Presents the parameter types as a list (a convenience method). The list will be immutable.

Specified by:

`parameterList` in interface `TypeDescriptor.OfMethod<Class<?>,MethodType>`

Returns:

the parameter types (as an immutable list)

lastParameterType

```
public Class<?> lastParameterType()
```

Returns the last parameter type of this method type. If this type has no parameters, the sentinel value `void.class` is returned instead.

API Note:

The sentinel value is chosen so that reflective queries can be made directly against the result value. The sentinel value cannot be confused with a real parameter, since `void` is never acceptable as a parameter type. For variable arity invocation modes, the expression `lastParameterType().getComponentType()` is useful to query the type of the "varargs" parameter.

Returns:

the last parameter type if any, else `void.class`

Since:

10

parameterArray

```
public Class<?>[] parameterArray()
```

Presents the parameter types as an array (a convenience method). Changes to the array will not result in changes to the type.

Specified by:

parameterArray in interface TypeDescriptor.OfMethod<Class<?>,MethodType>

Returns:

the parameter types (as a fresh copy if necessary)

equals

```
public boolean equals(Object x)
```

Compares the specified object with this type for equality. That is, it returns `true` if and only if the specified object is also a method type with exactly the same parameters and return type.

Overrides:

equals in class Object

Parameters:

x - object to compare

Returns:

`true` if this object is the same as the obj argument; `false` otherwise.

See Also:

Object.equals(Object)

hashCode

```
public int hashCode()
```

Returns the hash code value for this method type. It is defined to be the same as the hashcode of a List whose elements are the return type followed by the parameter types.

Overrides:

hashCode in class Object

Returns:

the hash code value for this method type

See Also:

Object.hashCode(), equals(Object), List.hashCode()

toString

```
public String toString()
```

Returns a string representation of the method type, of the form `"(PT0,PT1...)RT"`. The string representation of a method type is a parenthesis enclosed, comma separated list of type names, followed immediately by the return type.

Each type is represented by its `simple name`.

Overrides:

`toString` in class `Object`

Returns:

a string representation of the object.

fromMethodDescriptorString

```
public static MethodType fromMethodDescriptorString(String descriptor,
                                                    ClassLoader loader)
                                             throws
IllegalArgumentException,
TypeNotPresentException
```

Finds or creates an instance of a method type, given the spelling of its bytecode descriptor. Convenience method for `methodType`. Any class or interface name embedded in the descriptor string will be resolved by the given loader (or if it is null, on the system class loader).

Note that it is possible to encounter method types which cannot be constructed by this method, because their component types are not all reachable from a common class loader.

This method is included for the benefit of applications that must generate bytecodes that process method handles and `invokedynamic`.

Parameters:

`descriptor` - a bytecode-level type descriptor string "(T...)T"

`loader` - the class loader in which to look up the types

Returns:

a method type matching the bytecode-level type descriptor

Throws:

`NullPointerException` - if the string is null

`IllegalArgumentException` - if the string is not well-formed

`TypeNotPresentException` - if a named type cannot be found

`SecurityException` - if the security manager is present and `loader` is `null` and the caller does not have the `RuntimePermission("getClassLoader")`

toMethodDescriptorString

```
public String toMethodDescriptorString()
```

Returns a descriptor string for the method type. This method is equivalent to calling `MethodType::descriptorString`.

Note that this is not a strict inverse of `fromMethodDescriptorString`. Two distinct classes which share a common name but have different class loaders will appear identical when viewed within descriptor strings.

This method is included for the benefit of applications that must generate bytecodes that process method handles and `invokedynamic`. `fromMethodDescriptorString`, because the latter requires a suitable class loader argument.

Returns:

the descriptor string for this method type

See *Java Virtual Machine Specification*:

4.3.3 Method Descriptors

See Also:

Nominal Descriptor for `MethodType`

descriptorString

```
public String descriptorString()
```

Returns a descriptor string for this method type.

If this method type can be described nominally, then the result is a method type descriptor (JVMS 4.3.3). `MethodTypeDesc` for this method type can be produced by calling `MethodTypeDesc::ofDescriptor` with the result descriptor string.

If this method type cannot be described nominally and the result is a string of the form:

```
"(<parameter-descriptors>)<return-descriptor>"
```

where `<parameter-descriptors>` is the concatenation of the descriptor string of all of the parameter types and the descriptor string of the return type. No `MethodTypeDesc` can be produced from the result string.

Specified by:

`descriptorString` in interface `TypeDescriptor`

Returns:

the descriptor string for this method type

See *Java Virtual Machine Specification*:

4.3.3 Method Descriptors

Since:

12

See Also:

Nominal Descriptor for `MethodType`

describeConstable

```
public Optional<MethodTypeDesc> describeConstable()
```

Returns a nominal descriptor for this instance, if one can be constructed, or an empty `Optional` if one cannot be.

Specified by:

`describeConstable` in interface `Constable`

Returns:

An `Optional` containing the resulting nominal descriptor, or an empty `Optional` if one cannot be constructed.

Since:

12

See Also:

Nominal Descriptor for `MethodType`
