MethodHandles

An IBM Implementation

Dan Heidinga
J9 VM Software Developer
daniel_heidinga@ca.ibm.com

Disclaimer

- **Standard disclaimers apply:**
  - Implementation details may change
  - Talk addresses possible implementation, but doesn't guarantee this is what we'll do
  - This is a promise free talk – no shipping dates, no guarantee this implementation is what we'll use, etc

Agenda

- **MethodHandles overview**
  - Categories of MethodHandles
  - “Primitive” handle design
  - “Java” handle design
- **MethodHandle invocation**
  - invokehandle bytecode?
  - Turning “invokeGeneric” into an exact invoke
- **JIT tricks**

MethodHandles: JSR 292’s crown jewel

- **MethodHandle is the key to JSR 292**
  - The most fundamental change to Java since its inception
  - Enables invokedynamic and is a crucial building block for other projects
- **So what it is?**
  - A method pointer: invokestatic, invokevirtual, invokeinterface, invokespecial
  - A field pointer: getfield, putfield, getstatic, putstatic
  - An array accessor: aaload, aastore, caload, castore, etc
  - A constructor: new
  - A try{} catch{} block
  - An exception thrower: athrow
  - Stack manipulation
  - And more

Categories of MethodHandles

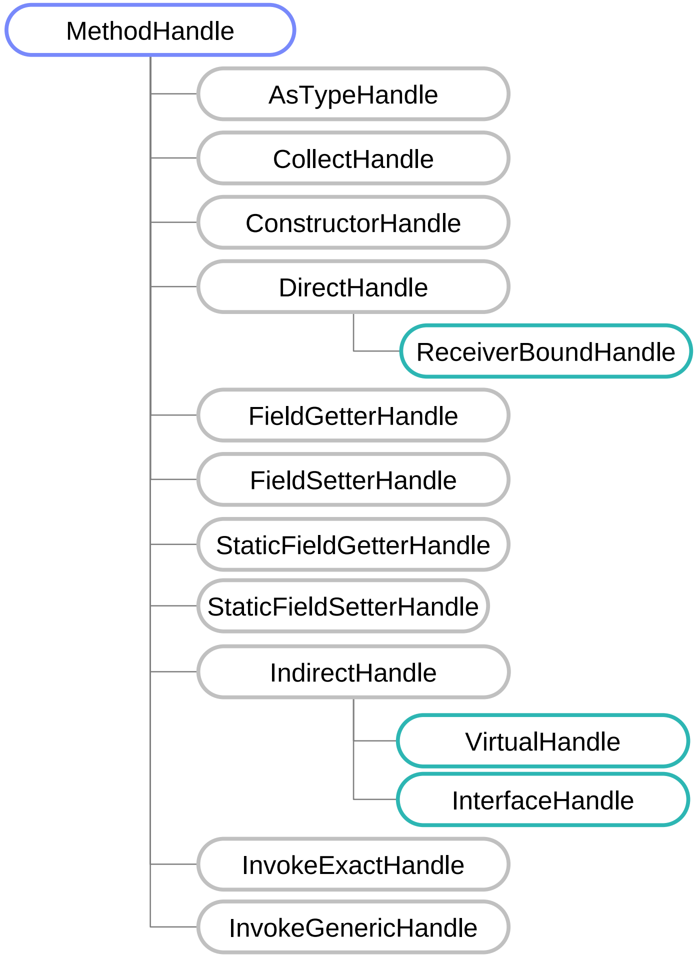
bind
findConstructor
findGetter(+ static)
findSetter (+ static)
findSpecial
findStatic
findVirtual (+ interface)

arrayElementGetter
arrayElementSetter
catchException

collectArguments / asCollector
convertArguments / asType

dropArguments
filterArguments
foldArguments
guardWithTest
insertArguments
permuteArguments
spreadArguments / asSpreader
throwException

Primitive handle type-hierarchy



MethodHandle.asType(…)

MethodHandle.asCollector(…)

MethodHandles.lookup().findConstructor(…)

MethodHandles.lookup().findStatic(…)

MethodHandles.lookup().findSpecial(…)

MethodHandles.lookup().bind(…)

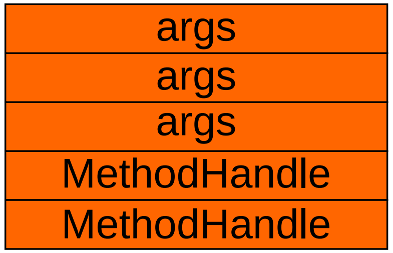
MethodHandles.lookup().findGetter(…)

MethodHandles.lookup().findSetter(…)

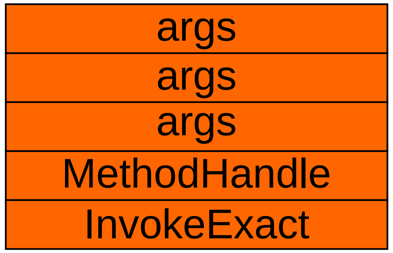
MethodHandles.lookup().findVirtual(…)

InvokeExact & InvokeGeneric special-cases

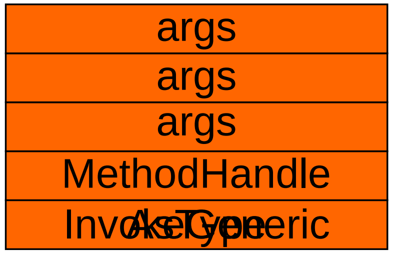
- No “real” method use in the dispatch
- InvokeExact
  - » slide the stack down by a slot and re-dispatch
- InvokeGeneric
  - » Play the same trick if the types are an exact match
  - » Replace the “original” MH with a new AsTypeHandle and re-dispatch



invokevirtual “invokeExact”,(Ljava/lang/MethodHandle;III)V

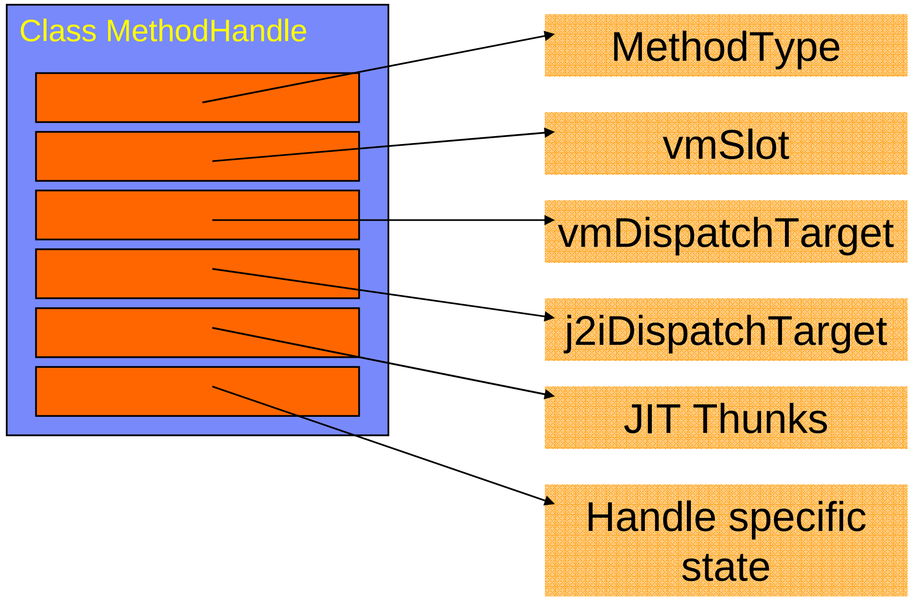


invokevirtual “invokeExact”,(III)V



invokevirtual “invokeExact”,
(Ljava/lang/MethodHandle;III)V

Primitive MethodHandle: Object layout



“Java” MethodHandles

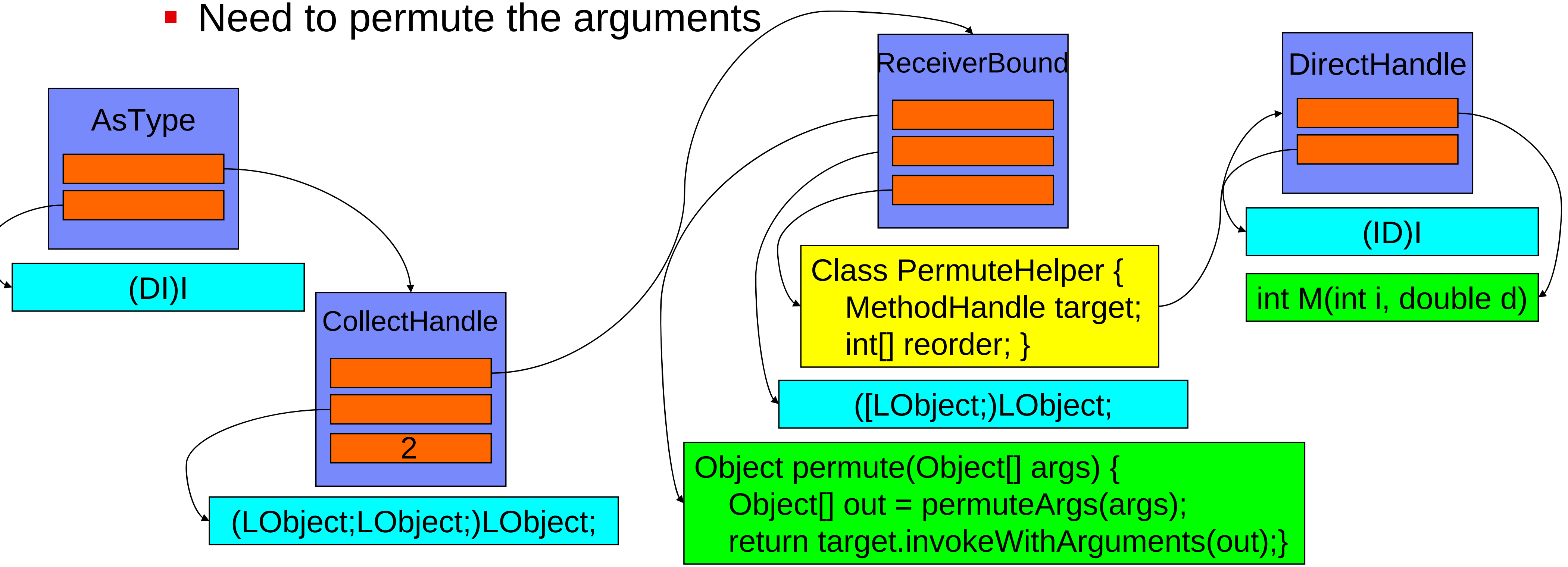
- The remaining “adapter” handles can be implemented in Java
- They fall into 2 categories:
  - » Simple – single method, possibly with an AsTypeHandle wrapper
  - » Complicated – Requires a chain of primitive handles to implement

Simple “Java” MethodHandles: ArrayElementGetter

```
public static MethodHandle arrayElementGetter(Class arrayType) {
        Class componentType = arrayType.getComponentType();
        if (componentType.isPrimitive()) {
                // Directly lookup the appropriate helper method
                String name = componentType.getCanonicalName();
                MethodType type = MethodType.methodType(componentType, arrayType, int.class);
                return lookup().findStatic(MethodHandles.class, name + "ArrayGetter", type);
        }
        // Lookup the "Object[]" case and use asTypes() to get the right MT and return type.
        MethodType type = MethodType.methodType(Object.class, Object[].class, int.class);
        MethodHandle mh = lookup().findStatic(MethodHandles.class, "objectArrayGetter", type);
        MethodType realType = MethodType.methodType(componentType, arrayType, int.class);
        return mh.asType(realType);
}
```

Complicated “Java” MethodHandles

- MethodHandle on a static method: int M(int i, double d)
- Want to call this as: int M_prime(double d, int i)
- Need to permute the arguments



```
Class PermuteHelper {
    MethodHandle target;
    int[] reorder; }
```

```
Object permute(Object[] args) {
    Object[] out = permuteArgs(args);
    return target.invokeWithArguments(out);}
```

MethodHandle invocation

- Two methods with “magic” abilities:
  public native @PolymorphicSignature <R,A> R invokeExact(A … args) throws Throwable
  public native @PolymorphicSignature <R,A> R invokeGeneric(A … args) throws Throwable

- Don’t appear in the classfile
  - » Normal method resolution can’t succeed
  - » Verification of them is meaningless (runtime)
  - » Impose a tax on invokevirtual

- We need more invoke bytecodes!
  - » invokehandle: MethodHandle.invokeExact
  - » invokehandlegeneric: MethodHandle.invokeGeneric

invokehandlegeneric: a late bound asType handle

- invokeGeneric:
  - » allows type conversions between the stack arguments and the MethodHandle’s expected type.
- MethodHandle.asType(MethodType):
  - » allows type conversions between the stack arguments and the target MethodHandle’s expected type
- Sounds very similar, doesn’t it?

invokehandlegeneric bytecode:

```
resolvedType <- cpEntry.MethodType
handleType <- receiver.MethodType
resolvedType == handleType
    run handle
else
    run handle.asType(resolvedType)
```

JIT Tricks

- **JSR 292 is a bit of a different approach for us**
  - Implemented in Java as much as possible
- **MethodHandle’s class hierarchy**
  - JIT can easily convert from Class to its Intermediate Language (IL)
- **JIT thunks: thunk_exact & thunk_generic**
  - Per-signature optimized code sequence to call the handle
    - » Some are per-signature + data
  - Shared across different kinds of handles
  - Per-instance thunks are created when a MethodHandle chain is compiled into a single code block

Conclusion

- MethodHandles are a powerful fusion of VM+JIT
  - » JSR 292
  - » Re-implement java.lang.Reflect?
- Performance for ***your*** language
- Future directions
  - » More optimizations!

Legal Notices