Lambdas Internals and In Practice

Vaibhav Choudhary (@vaibhav_c)
Java Platforms Team
http://blogs.oracle.com/vaibhav

Agenda

- Lambda Basics
- Bytecode and InvokeDynamic (Indy)
- Lambda approach and implementation
- Lambda performance evaluation
- Lambda Utilities
- QA + Resources

Lambda Basics

- A lambda expression is like “anonymous method”.
  - Has an argument list, return, and a body
  - `people.forEach((Person e) -> names.add(e.getName));`
- Can capture value from enclosing context
  - `people.removeAll(e -> e.getAge() < minAge);`
- Based on Lambda calculus
  - `(λ x. x+10)5 => 15`
  - `(λ x.(λ y. x+y))(5) => λy.5+y`

Why we are doing it

- A better respect to multi-core
- Cleaner way to write parallel programming
- Empower Library Developers.
- Focus on “what” to do rather than “how” to do.
- We want to get rid of inner classes.

- How to represent lambda ?

Representation of Lambda

- Java is a typed language and the first question that comes - what is the type of Lambda.
- Java has no function type like other languages and VM don’t have any representation for function type.
- Adding function type is a option but
    - How the signature looks like ?
    - What bytecode call it will make ?
    - How to deal with different variance?
- Minimal change at VM and something fast would help us.

Representation of Lambda

- Basically we have to represent a function, a kind of single method.
- Historically we are using Interfaces to do so like Comparator, Runnable and many more.
- So, why not use Interfaces and tell them "Functional Interfaces" (Lets not complicate with new features)

```
people.removeAll(e -> e.getAge() < minAge);

interface Predicate <Person> { boolean test(Person P); }
    Interface Predicate <T> { boolean test(T t); }
```

Representation of Lambda

- Alright, so now how to create instance of it ?
- Lets see, some of the approaches that can be thought of !
- Idea is
  - Faster
  - Less complicated
  - Clear at API level
- These changes will last forever and we can’t change the representation in the next release.

Representation of Lambda – A1

- Inner classes

```
people.removeAll(e -> e.getAge() < minAge);

    class Inner$1 implements Predicate <Person> {
            private int $a0; // for minAge
          Inner$1(int a0) { this.$a0 = a0; }
public boolean test (Person p) { return p.age < $a0; }
```

Representation of Lambda – A1

- Issues with Inner class implementation
  - I contradicted.
  - Performance issues :-
    - We don't want to inherit the problems of inner class
    - One inner class per lambda.
  - This will become binary representation forever

So, lets think of second approach …

Representation of Lambda – A2

- JDK7 offers MethodHandle.
- Lets see if we can use MethodHandle for our current problem.
  - Can store reference of method in constant pool.
  - Can obtain method handle for any method.
  - VM can inline it.
  - Some of the demo.
  - Actually it can do anything.
  - Some demo, how method handle work.
- Translate lambda (language-level) into MethodHandle (VM level)

Representation of Lambda – A2

```
list.removeAll(p -> p.getAge() >= minAge);
```

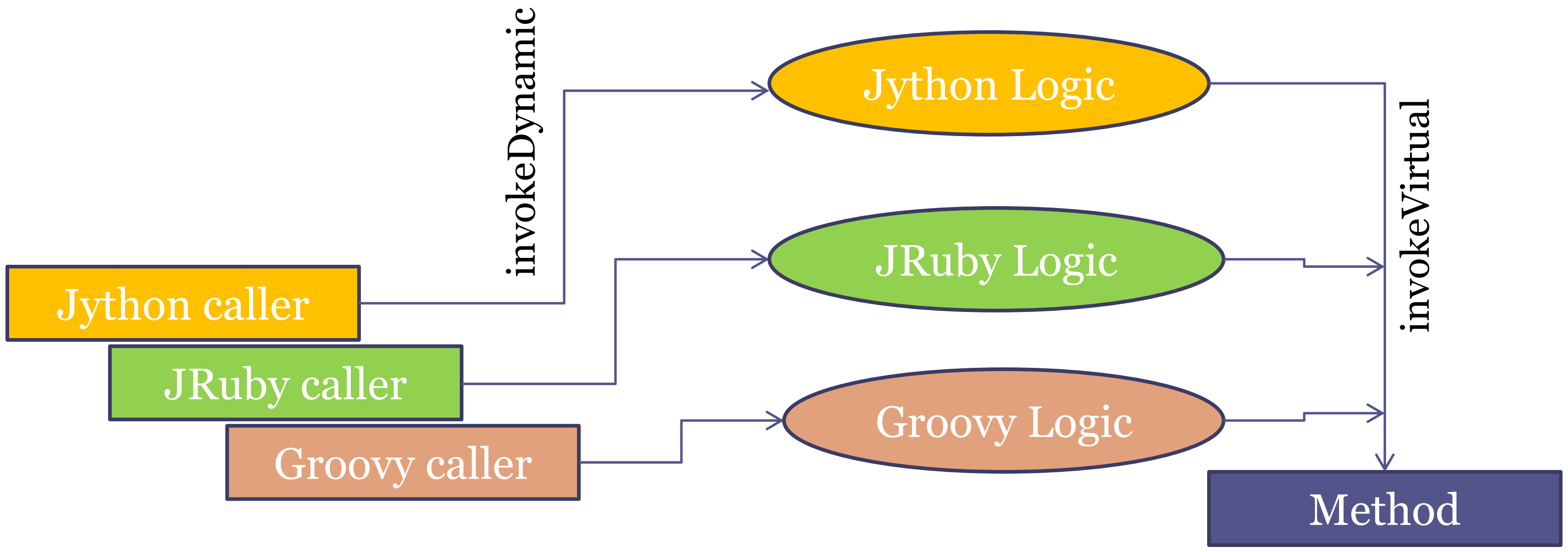
```
MethodHandle mh = LDC[lambda$1];
mh = MethodHandles.insertArguments(mh,0, minAge);
list.removeAll(mh);

private static boolean lambda$1(int capture, Person p) {
return p.getAge() >= capture; }
```

Representation of Lambda – A2

- Signature of List.removeAll would be :-
  - void removeAll(MethodHandle predicate)
- Overloading an issue ?
- Yuck !!
- Change forever – No.
- Good for runtime but at language level it is not possible.

Representation of Lambda – A3

- JDK7 provides another feature called "InvokeDynamic"
- InvokeDynamic is a new bytecode change.



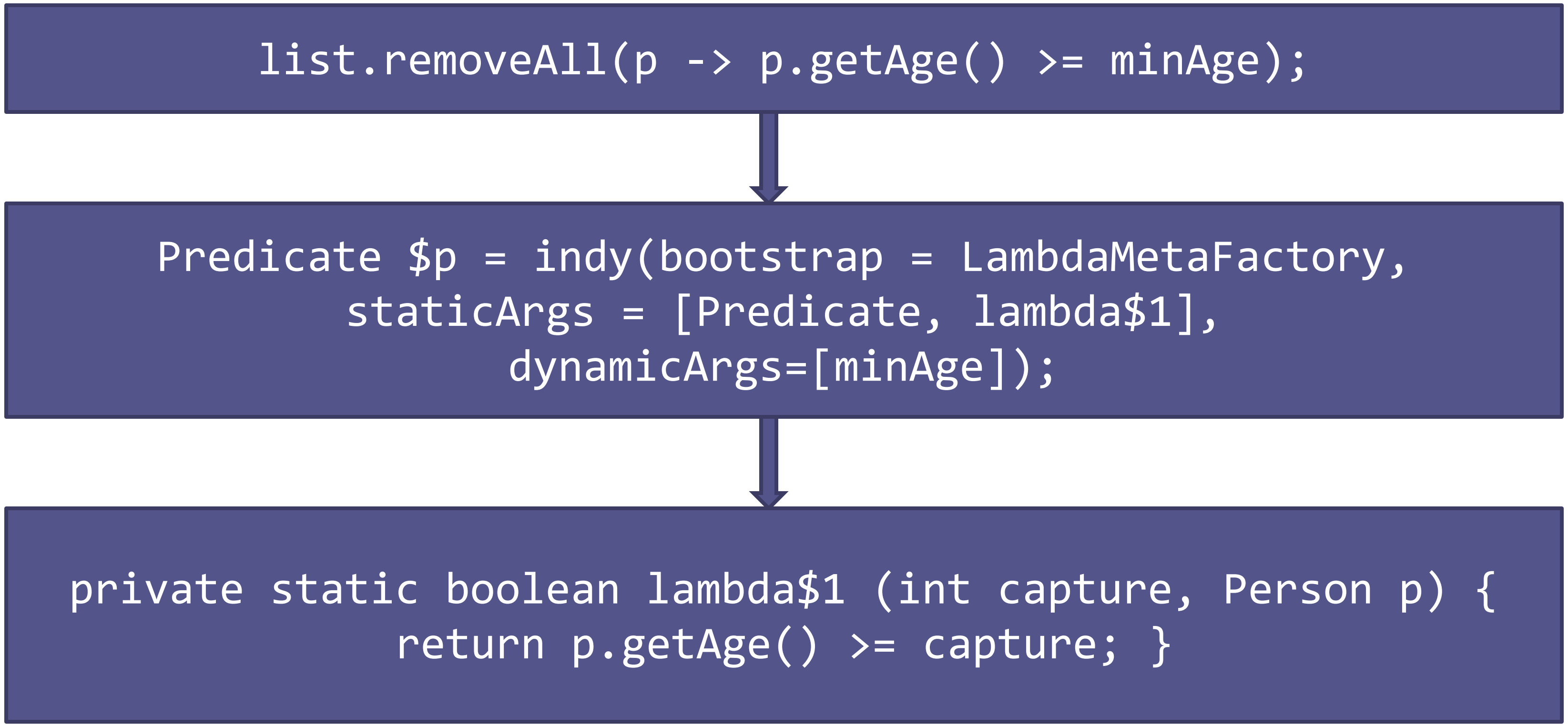
Invocation at Bytecode Level

- Invokestatic – static methods
- Invokevirtual – class methods
- Invokeinterface – interface methods
- Invokespecial – private methods/constructors
- Invokedynamic – why ?

Representation of Lambda – A3

- invokeDynamic at VM level and functional interface at Language level.
- def add (a, b) { a + b }
  - If language support integer and float. Then, what can be at runtime
  - Consult bootstrap at the first time.
  - Bootstrap will pass it to the “call site”.
  - Call Site will have logic for resolution and also if it fails (re-link logic)
  - Preventive vs Practical

Representation of Lambda – A3



```
list.removeAll(p -> p.getAge() >= minAge);
```

```
Predicate $p = indy(bootstrap = LambdaMetaFactory,
staticArgs = [Predicate, lambda$1],
dynamicArgs=[minAge]);
```

```
private static boolean lambda$1 (int capture, Person p) {
return p.getAge() >= capture; }
```

Indy Structure

-
  ```
  CONSTANT_InvokeDynamic_info {
         u1 tag;
         u2 bootstrap_method_attr_index;
         u2 name_and_type_index;
         }
  ```

- Please see JLS for more details.

Invokedynamic Invocation (Indy)

- JSR – 292
  - java.lang.invoke API
    - MethodHandle
    - CallSite
    - BootStrap Method
  - InvokeDynmaic

Lambda Performance

- A comparison with Anonymous class.
- Capturing and Non-capturing Lambdas.

Lambda	Anonymous Class
Linkage	Class Loading
Capture	Instantiation
Invocation	Invocation

Lambda – Linkage Benchmarking

```
@FunctionalInterface public interface Level { Level up (); }
public static Level get1023 ( String p) {
      return () -> get1022 (p);
}
public static Level get1024 ( String p) {
      return () -> get1023 (p);
}

public static Level get1024 ( final String p) {
 return new Level () {
 @Override public Level up ()
 { return get1023 (p); }};}
```

Lambda – Linkage Benchmarking

```
public static Level get1023 ( String p) {
  return () -> get1022 (p);
  }
  public static Level get1024 ( String p) {
  return () -> get1023 (p);
  }
…….

Chain of calls :-
()->()->()->()-> …
```

Lambda – Linkage Result (Cold)

	Anonymous (-TC)	Lambda (-TC)	Anonymous (+TC)	Lambda (+TC)
1K	7.24	0.95	6.98	0.77
4K	16.64	2.46	16.16	1.84
16K	22.44	5.92	21.25	4.90
64K	34.52	18.20	33.34	16.33

-What do you think of Hot Result ?
-Excellent performance achievement.
-Major contributor: 25% resolve_indy, 13% link_MH_constant, 44%LambdaMetaFactory, 20% Unsafe.defineClass

Non-Capture Lambda: Benchmarking

-
  ```
  public static Supplier < String > lambda () {
      return () -> "42";
  }
  ```

-
  ```
  public static Supplier < String > anonymous () {
      return new Supplier < String >() {
          @Override
          public String get () {
              return "42";
          }
      };
  }
  ```

Non-Capturing Lambda - Results

	Single thread	Multi-threading (MAX=4)
anonymous	6.02 ± 0.02	12.40 ± 0.09
cached anonymous	5.36 ± 0.01	5.97 ± 0.03
Lambda	5.31 ± 0.02	5.93 ± 0.07

Capture Lambda: Benchmarking

-
  ```
  public Supplier < String > lambda () {
          String localString = someString ;
          return () -> localString ;
  }
  ```

-
  ```
  public static Supplier<String>anonymous() {
          String localString=someString;
          return new Supplier<String>() {
  @Override
  public String get() {
          return localString;
  }
          };
  }
  ```

Capturing Lambda - Results

	Single thread	Multi-threading (MAX=4)
anonymous(static)	6.94 ± 0.03	13.4 ± 0.33
anonymous(non-static)	7.88 ± 0.09	18.7 ± 0.17
Lambda	8.29 ± 0.04	16.0 ± 0.28

Lambdas Usages

- Plug new feature, new condition
  - CashCounter
  - SocialNetworkBuilder
- Move existing code to Lambdas for performance.
- Parallel Computing – Not a developer’s headache.
- Focus on “what to do” not on “how to do”.

Q/A : Some of useful resources

- JavaOne 2015 Channel: https://www.youtube.com/channel/UCdDhYMT2USoLdh4SZIsu_1g
- JVM support for Non-Java Language: http://docs.oracle.com/javase/7/docs/technotes/guides/vm/multiple-language-support.html
- Lambda Expressions: https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html
- My blogs: http://blogs.oracle.com/vaibhav