

[www.qconferences.com](http://www.qconferences.com)  
[www.qconbeijing.com](http://www.qconbeijing.com)



QCon北京2014大会 4月17—19日

伦敦 | 北京 | 东京 | 纽约 | 圣保罗 | 上海 | 旧金山

London · Beijing · Tokyo · New York · Sao Paulo · Shanghai · San Francisco

# QCon全球软件开发大会

International Software Development Conference



@InfoQ



infoqchina

软件  
正在改变世界！

# Introducing Project Lambda and Invokedynamic

Lambda is coming, are you ready?

# Agenda

- Knowing Lambda
- Typing
- Invokedynamic
- Lambda at Runtime

# About Me

- 叶瑞华, Roger, @ryenus
- A Java developer & tech lead
- <http://osrc.dfm.io/ryenus>  
(try it!)



# About You?



- Should be using or used Java, right?
- Anyone used tools like javap?
- Anyone tried Scala, Groovy or ...?
- Why?

# An Inconvenient Truth about Java

- new Thread(new Runnable() {  
    @Override  
    public void run() {  
        System.out.println(42);  
    }  
});

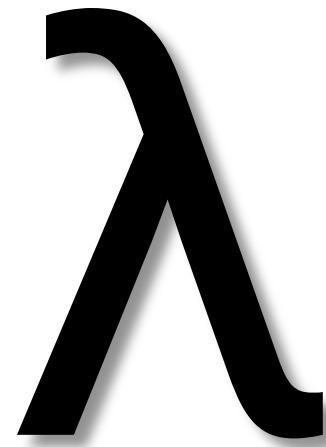
# An Inconvenient Truth about Java

- ```
List<String> names = Arrays.asList(...);
List<String> upperNames = new
ArrayList<String>();

for (String name : names) {
    upperNames.add(
        name.toUpperCase());
}
```

# Introducing Lambda

- (a, b) -> a \* b
- x -> x + 2
- () -> 42
- (String s) -> System.out.println(x)
- list -> { list.add(42); return list; }

A large, bold, black Greek letter lambda (λ) is positioned on the right side of the slide. It is oriented vertically and has a slight shadow, giving it a three-dimensional appearance.

# Using Lambda

- new Thread(  
    () ->  
    System.out.println(42);  
);

# More Lambda Use Cases

- `Collections.sort(words,  
              (w1, w2) -> w1.length() - w2.length());`
- `words.forEach(w -> System.out.println(x));`
- `numbers.stream().filter(n -> n > 0);`
- `words.stream().parallel().map(w ->  
                          w.toUpperCase()));`
- `numbers.stream().reduce((s, n) -> s + n);`

# Lambda Benefits

- More Expressive API:
  - `forEach`, `filter`, `map`, `reduce`
- Passing code as data, think about Visitor pattern
- Parallel almost free, as in
  - `words.stream().parallel().map(...)`
  - more about this later ...

# Gotcha of ‘this’ Keyword

```
void checkThis() {  
    System.out.println("outer " + this); // A  
    new Thread(new Runnable() {  
        @Override  
        public void run() {  
            System.out.println("inner " + this); // B  
            printThis(); // X?  
        }  
    }).start();  
  
    new Thread(() ->  
        System.out.println(this + " in lambda")).start(); // Y?  
}  
  
void printThis() { System.out.println(this + " from printThis()"); }
```

# Lambda Scope and Naming

- Lambdas are in the same scope as the enclosing context
  - **this** refers to the enclosing class
  - Lambda parameters cannot shadow outer local variables

# Typing

# Type of Lambdas

- Are Lambdas Objects?
- Yes, but you **CANNOT** do:  
 `Object o = x -> System.out.println(x);`
- However you CAN do:  
 `Consumer<?> c = x -> System.out.println(x);`  
`Object obj = c;`

# Functional Interfaces

- Definition

```
@FunctionalInterface  
public interface Consumer<T> {  
    void accept(T t);
```

- Invocation

```
public interface Iterable<T> {  
    Iterator<T> iterator();  
    default void forEach(Consumer<? super T> action) {  
        Objects.requireNonNull(action);  
        for (T t : this) {  
            action.accept(t);  
        }  
    }
```

# Built-in @FunctionalInterface(s)

From built-in package: `java.util.function`

- |             |               |                   |
|-------------|---------------|-------------------|
| • Consumer  | • BiConsumer  |                   |
| • Function  | • BiFunction  | • BinaryOperator  |
| • Predicate | • BiPredicate |                   |
| • Supplier  |               | • BooleanSupplier |

Variations:

- |                       |                                                                    |
|-----------------------|--------------------------------------------------------------------|
| • Number of Arguments | Function, Supplier                                                 |
| • Argument Types      | BiFunction, BinaryOperator                                         |
| • Return Types        | Consumer, Function, Predicate,<br>UnaryOperator, IntToLongFunction |

\* You can also build your own Functional Interfaces

# Default Methods

- Expanding interfaces without breaking existing code

```
public interface Iterable<T> {  
  
    Iterator<T> iterator();  
  
    default void forEach (Consumer<? super T> action) {  
        Objects.requireNonNull(action);  
        for (T t : this) {  
            action.accept(t);  
        }  
    }  
}
```

# Multi-Inheritance

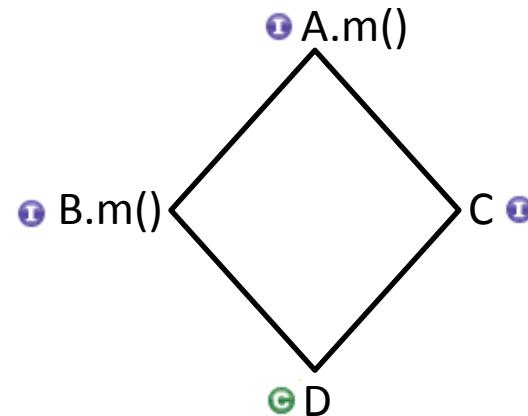
- Question: Do we have a conflict here?

```
interface A {  
    default void m() { print("A"); }  
}
```

```
interface B extends A {  
    default void m() { print("B"); }  
}
```

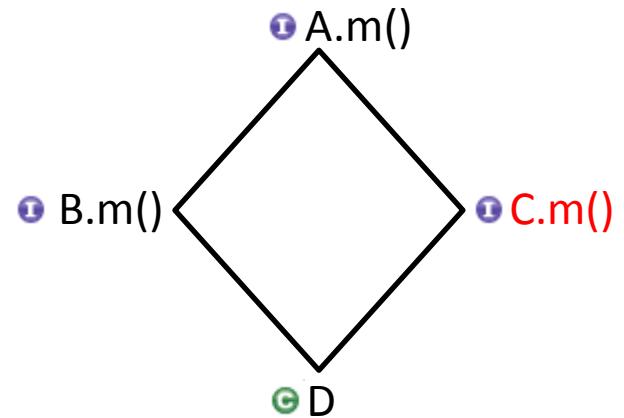
```
interface C extends A { }  
class D implements B, C { }
```

```
C c = new D();  
c.m(); // A? or B?
```



# Resolving Conflict

```
interface A {  
    default void m() { print("A"); }  
}  
interface B extends A {  
    default void m() { print("B"); }  
}  
interface C extends A {  
    default void m() { print("C"); }  
}  
//class D implements B,C { } //Error: duplicate default methods  
  
class D implements B, C {  
    public void m() { C.super.m(); } // new syntax!  
}
```



# Lambda vs. Existing Method

- Lambda
  - $x \rightarrow x + 2$
- Method:
  - `class C { int plus2() { return x + 2; }}`
- Question:
  - What information does `C.plus2` carry?
  - How about comparing it to the lambda expression?

# Method References

- Method references are interchangeable with equal lambda expressions

```
List<Integer> numbers = Arrays.asList(1, 2, 3);
```

```
// static method, as parameter  
numbers.stream().map(String::valueOf)
```

```
// instance method (PrintStream::println), as parameter  
numbers.forEach(System.out::println)
```

```
// instance method, as receiver  
numbers.stream().map(Number::longValue)
```

```
// virtual method, as receiver  
numbers.sort(Comparable::compareTo)
```

# Recursive Lambda Definition?

- ```
UnaryOperator<Integer> fib =  
    n -> n < 2 ? 1  
        : fib.apply(n - 1) + fib.apply(n - 2);
```

# Lambda vs. Inner Class – Under the Cover

- The Source Code

```
List<String> words = Arrays.asList("Tom", "Alex");  
  
words.sort((a, b) -> a.length() - b.length());  
  
words.sort(new Comparator<String>() {  
    @Override  
    public int compare(String a, String b) {  
        return a.length() - b.length();  
    }  
});
```

# Lambda vs. Inner Class – Under the Cover

- The Byte Code

InnerClasses:

#8; //class Words\$1

BootstrapMethods:

0: #37 invokestatic

java/lang/invoke/LambdaMetafactory.metafactory:(Lookup; String;  
MethodType; MethodType; MethodHandle; MethodType;) CallSite;

14: invokestatic #5 // Method java/util/Arrays.asList

19: invokedynamic #6, 0 // InvokeDynamic  
#0:compare:()Ljava/util/Comparator;

24: invokeinterface #7, 2 // InterfaceMethod java/util/List.sort

35: invokespecial #9 // Method Words\$1."<init>":(LWords;)V

38: invokeinterface #7, 2 // InterfaceMethod java/util/List.sort

# invokedynamic

# Existing Invocation Instructions

- `Invokestatic`: for static methods
- `Invokevirtual`: for class methods
- `Invokeinterface`: for interface methods
- `Invokespecial`: for private methods, constructors, super-calls

# JVM as a Language Platform

- It's now hosts a lot of other language implementations
  - JavaScript ([Nashorn](#), [dynjs](#))
  - Ruby ([JRuby](#))
  - ...
- But
  - JavaScript has prototype based class inheritance
  - Ruby has singleton class

# Introducing Invokedynamic

- Invokedynamic is used to make a recipe to construct a lambda instance, including:
  - the functional interface to apply
  - the implementation method
  - the captured values
- The capture site is called the lambda factory
  - invoked with invokedynamic, return an instance of the Functional Interface
  - invoked only once, bypassed by future invocation

# The Lambda Metafactory

- Invokedynamic is used to make a recipe to construct a lambda instance, including:

```
metaFactory(MethodHandles.Lookup caller, // provided by VM
            String invokedName,          // provided by VM
            MethodType invokedType,       // provided by VM
            MethodHandle descriptor,      // lambda descriptor
            MethodHandle impl)           // lambda body
```

# Lambda at Runtime

# Translation Strategies

- Lambda is at least as fast as Inner Class
- Definitely could/would be faster in the future

# Lambda vs. Inner Class - Performance

- Lambda is at least as fast as Inner Class
- Definitely could/would be faster in the future

# IDE Support and Gotchas

- IntelliJ IDEA and Netbeans works
- Eclipse Luna early builds available, buggy per my experience

Questions?

Q&A

# References

- OpenJDK: Project Lambda
  - <http://openjdk.java.net/projects/lambda/>
- Lambda FAQ
  - <http://www.lambdafaq.org/>
- First Taste of Invokedynamic
  - <http://blog.headius.com/2008/09/first-taste-of-invokedynamic.html>
- Translation of Lambda Expressions
  - <http://cr.openjdk.java.net/~briangoetz/lambda/lambda-translation.html>
- Lambda: A Peek Under the Hood
  - <http://www.slideshare.net/jaxlondon2012/lambda-a-peek-under-the-hood-brian-goetz>
  - [http://www.youtube.com/watch?v=C\\_QbkGU\\_IqY](http://www.youtube.com/watch?v=C_QbkGU_IqY)



# Into the Bright Future



The Eight Works  
DIGITAL LAB

# 特别感谢 QCon上海合作伙伴

