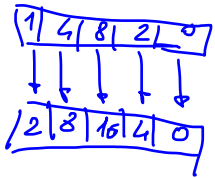
Lambda Expressions in Java

8 (2014)



Marco Alberti

Università degli Studi di Ferrara

Ingegneria del Software Avanzata

CdLM in Ingegneria Informatica e dell'Automazione

A.A. 2021-2022

Ultima modifica: 3 aprile 2022

Sommario

- Esempio
- Definizione generale
- Lambda expressions in Java

Lambda expression

Origine: λ-Calculus (Church, Anni 30), formalismo di logica matematica equivalente alle macchine di Turing

Sintassi astrazione

$\langle \lambda\text{-}expression \rangle ::= \lambda \langle parametro \rangle \ . \ \langle espressione \rangle$

Questo costrutto viene detto abstraction in λ-calculus e rappresenta la funzione che mappa il parametro nel valore rappresentato dall'espressione

Esempio

$\lambda x.2x$ è la funzione che calcola il doppio del suo parametro

Lambda expressions a più parametri

L'espressione di una lambda expression può a sua volta essere una lambda expression. Questo consente di rappresentare funzioni con più di un parametro.

Esempio

$\lambda x.(\lambda y.(x + y))$ rappresenta

- la funzione che a un valore x associa la funzione che a un valore y associa la somma di x e y
- cioè la funzione che a x e y associa la loro somma.

Applicazione

Le funzioni danno valori se applicate a parametri.
La sintassi dell'applicazione in λ-calculus è la semplice giustapposizione.

Sintassi applicazione

$\langle application \rangle ::= \langle expression \rangle \langle expression \rangle$

Esempio

$(\lambda x.2x)(5)$ vale 10.

Esempio

$(\lambda y.(\lambda x.(x + y)))(5)(4)$ vale 9.

Lambda expressions nei linguaggi di programmazione

- Da sempre presenti nei linguaggi funzionali
  - Lisp: Lisp 1.5, Scheme, Common Lisp, Clojure (tipizzati dinamicamente)
  - ML: SML, Ocaml, F#, Haskell (tipizzati staticamente)
- Introdotte nei linguaggi imperativi
  - C++ (dalla versione C++11; C ha solo puntatori a funzione)
  - Linguaggi di scripting (Python, Ruby, Perl)
  - JavaScript
  - C# (dalla versione 3, 2008)
  - Java (dalla versione 8, 2014)

Lambda expressions in Java

- Introdotte in Java 8
- come costrutto sintattico per l'implementazione di interfacce funzionali
- Un'interfaccia funzionale è un'interfaccia con un solo membro astratto.
- Alcune interfacce funzionali predefinite (nel package `java.util.function`):

Interfaccia	Tipo argomento	Tipo ritorno	Metodo astratto
`Function<T1,T2>`	`T1`	`T2`	`T2 apply(T1 arg)`
`Consumer<T>`	`T`	-	`void consume(T arg)`
`Supplier<T>`	-	`T`	`T get()`

- Molte altre: `https://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html`

Implementazione di interfacce funzionali

Un'interfaccia funzionale può essere implementata implicitamente da

- un metodo
- una lambda expression di signature corrispondente al membro astratto

In particolare, il metodo o la lambda expression possono

- essere assegnate a un variabile dichiarata di tipo interfaccia funzionale
- passati come parametri di tipo interfaccia funzionale
- restituiti come valore di ritorno di un metodo che ha come tipo di ritorno un'interfaccia funzionale

Interfaccia g(...) {

Interfaccia a = ... → ..

return ... → ... ;

f(Interfaccia a){

f(... → ...);

}

Assegnamento a interfaccia funzionale: metodo definito

350_lambda_expressions_in_java/SupplierMetodo.java

```
1  import java.util.function.Supplier;
2
3  public class SupplierMetodo {
4
5      static Integer Cinque() {
6          return 5;
7      }
8
9      public static void main(String[] argv) {
10         Supplier<Integer> f;
11         f = SupplierMetodo::Cinque;
12         System.out.println(f.get());
13     }
14 }
```

5

Lambda expression in Java

Sintassi

⟨*lambdaExpression*⟩ ::= ⟨*parametri*⟩ `->` ⟨*corpo*⟩

⟨*corpo*⟩ ::= ⟦ ⟨*espressione*⟩ |`{` ⟦ ⟨*istruzione*⟩ ⟧*`}` ⟧

⟨*parametri*⟩ ::= ⟦ ⟨*identificatore*⟩ |⟨*listaParametri*⟩ ⟧

⟨*listaParametri*⟩ ::= `(` ⟦ ⟨*parametro*⟩ ⟦ `,` ⟨*parametro*⟩ ⟧*⟧`)`

⟨*parametro*⟩ ::= ⟦ ⟨*identificatore*⟩ ⟧⟨*identificatore*⟩

Esempio

- `() -> 1`
- `x -> x + 1` oppure `(int x) -> x + 1`
- `(x, y) -> x + y`
- `(int x, int y) -> {int a; a = x + y; return a;}`

Assegnamento a interfaccia funzionale: lambda expression

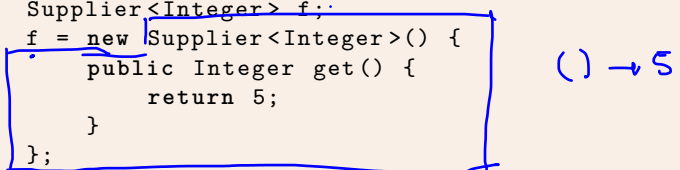
350_lambda_expressions_in_java/SupplierLambda.java

```
1  import java.util.function.Supplier;
2
3  public class SupplierLambda {
4      public static void main(String[] argv) {
5          Supplier<Integer> f;
6          f = () -> 5;
7          System.out.println(f.get());
8      }
9  }
```

Fino a Java 7: implementazione con classe anonima

350_lambda_expressions_in_java/SupplierClasseAnonima.java

```
1  import java.util.function.Supplier;
2
3  public class SupplierClasseAnonima {
4      public static void main(String[] argv) {
5          Supplier<Integer> f;
6          f = new Supplier<Integer>() {
7              public Integer get() {
8                  return 5;
9              }
10         };
11         System.out.println(f.get());
12         System.out.println(f.get());
13     }
14 }
```



Lambda expression come parametro

350_lambda_expressions_in_java/ForEach.java

```
1  import java.util.Arrays;
2  import java.util.List;
3  import java.util.function.Consumer;
4
5  public class ForEach {
6      public static <T> void perOgni(List<T> l, Consumer<T> azione) {
7          for (T e : l)
8              azione.accept(e);
9      }
10
11     public static void main(String[] argv){
12         List<String> l = Arrays.asList(argv);
13         perOgni(l, arg -> {
14             System.out.print(arg); System.out.print(";");
15         });
16         System.out.println();
17     }
18 }
```

Lambda expression come valore di ritorno

350_lambda_expressions_in_java/Ritorno.java

```
1  import java.util.function.Function;
2
3  public class Ritorno {
4      static Function<Integer, Integer> creaDoppio() {
5          return x -> 2 * x;
6      }
7
8      public static void main(String[] argv) {
9          Function<Integer, Integer> doppio = creaDoppio();
10         System.out.println(doppio.apply(5));
11     }
12 }
```

Lambda expression a due parametri

350_lambda_expressions_in_java/FunzioneDueParametri.java

```
1 public interface FunzioneDueParametri<T1,T2,T3> {
2     T3 applica(T1 par1, T2 par2);
3 }
```

350_lambda_expressions_in_java/DueParametri.java

```
1 public class DueParametri {
2     public static void main(String[] argv){
3         FunzioneDueParametri<Float, Integer, Float> f =
4             (Float x, Integer n) -> x * n;
5         System.out.println(f.applica(2.5f,3));
6     }
7 }
```

Closure in Javascript

350_lambda_expressions_in_java/counter.js

```
1  function makeCounter(init){
2      return () => {
3          init++;
4          return init-1;
5      }
6  }
7
8  c1 = makeCounter(1);
9  c2 = makeCounter(1);
10
11 console.log(c1()); 1
12 console.log(c1()); 2
13 console.log(c2()); 1
14 console.log(c2()); 2
```

init ∈ ENVIRONMENT + CODICE } CLOSURE

Stessa cosa in Java

350_lambda_expressions_in_java/Counter.java

```
1  import java.util.function.Supplier;
2
3  public class Counter {
4      static Supplier<Integer> makeCounter(int init){
5          return () -> {
6              init++; /* errore:
7              Local variable init defined in an enclosing scope
8              must be final or effectively final
9              */
10             return init-1;
11         };
12     }
13     public static void main(String[] argv){
14         Supplier<Integer> c1, c2;
15         c1 = makeCounter(1); c2 = makeCounter(1);
16         System.out.println(c1.get()); System.out.println(c1.get());
17         System.out.println(c2.get()); System.out.println(c2.get());
18     }
19 }
```

Cattura di membro statico

350_lambda_expressions_in_java/CounterStatico.java

```
1  import java.util.function.Supplier;
2
3  public class CounterStatico {
4      static int init = 1;
5
6      static Supplier<Integer> makeCounter(){
7          return () -> {
8              init++; // corretto: ma init e` condivisa!
9              return init-1;
10         };
11     }
12     public static void main(String[] argv){
13         Supplier<Integer> c1, c2;
14         c1 = makeCounter(); c2 = makeCounter();
15         System.out.println(c1.get()); System.out.println(c1.get());
16         System.out.println(c2.get()); System.out.println(c2.get());
17     }
18 }
```

Cattura variabili in lambda expression

In Java le lambda expression possono catturare

- variabili locali solo se final o effectively final
- variabili statiche e di istanza (che vengono condivise fra le funzioni anonime)

Discussione:
`https://www.bruceeckel.com/2015/10/17/are-java-8-lambdas-closures/`
Conclusione:

- L'introduzione delle lambda expressions non ha fatto di Java un linguaggio funzionale
- In molte occasioni, le lambda expression si dimostrano comunque un costrutto utile a semplificare il codice.

Contatori in C#

350_lambda_expressions_in_java/Counter.cs

```
using System;
public class Counter {
    static Func<int> makeCounter(int init){
        return () => {
            init++;
            return init-1;
        };
    }
    public static void Main(){
        Func<int> c1, c2;
        c1 = makeCounter(1); c2 = makeCounter(1);
        Console.WriteLine(c1()); Console.WriteLine(c1());
        Console.WriteLine(c2()); Console.WriteLine(c2());
    }
}
```