Java 8 引入了几种新的语言特性，旨在实现更快、更清晰的编码。 作为最重要的特性，即所谓的“Lambda 表达式”，它开启了函数式编程的大门。Lambda 表达式允许以简捷的方式实现和传递函数，而无需声明额外的（匿名）类。

### Flatmap

env.fromElements(1, 2, 3)

// 返回 i 的平方

.map(i -> i\*i)

.print();

由于 OUT 是 Integer 而不是泛型，Flink 可以由方法签名 OUT map(IN value) 的实现中自动提取出结果的类型信息。

不幸的是，flatMap() 这样的函数，它的签名 void flatMap(IN value, Collector<OUT> out) 被 Java 编译器编译为 void flatMap(IN value, Collector out)。这样 Flink 就无法自动推断输出的类型信息了。

Flink 很可能抛出类似如下的异常

**在这种情况下，需要 显式 指定类型信息，否则输出将被视为 Object 类型，这会导致低效的序列化。**

**例子 1：**

**package** com.sq3;  
  
**import** org.apache.flink.api.common.typeinfo.Types;  
**import** org.apache.flink.streaming.api.datastream.DataStream;  
**import** org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;  
**import** org.apache.flink.util.Collector;  
  
**public class** lam1 {  
 **public static void** main(String[] args) **throws** Exception {  
 StreamExecutionEnvironment env = StreamExecutionEnvironment.*getExecutionEnvironment*();  
 DataStream<Integer> input = env.fromElements(1,2,3);  
 input.flatMap((Integer number, Collector<String> out)->{  
 StringBuilder builder = **new** StringBuilder();  
 **for**(**int** i = 0; i < number; i++) {  
 builder.append(**"a"**);  
 out.collect(builder.toString());  
 }  
 }).returns(Types.***STRING***).print();  
 *//returns显式提供类型信息,print 打印 "a", "a", "aa", "a", "aa", "aaa"* env.execute(**"lam1"**);  
 }  
}

例子2：

**package** com.sq3;  
  
**import** org.apache.flink.api.common.typeinfo.Types;  
**import** org.apache.flink.streaming.api.datastream.DataStream;  
**import** org.apache.flink.streaming.api.datastream.DataStreamSink;  
**import** org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;  
**import** org.apache.flink.util.Collector;  
  
**public class** lam2 {  
 **public static void** main(String[] args) **throws** Exception {  
 StreamExecutionEnvironment env = StreamExecutionEnvironment.*getExecutionEnvironment*();  
 DataStream<String> input = env.fromElements(**"hello flink"**,**"hello spark"**);  
 DataStreamSink<String> inputw = input.flatMap((String i, Collector<String> out) -> {  
 String[] a = i.split(**" "**);  
 **for**(String word:a){  
 out.collect(word);  
 }  
 }).returns(Types.***STRING***).print();  
  
 env.execute(**"af"**);  
 }  
}

[hadoop@h201 flink-1.7.2]$ bin/flink run /home/hadoop/qq/f1flink12.jar --class com.sq3.lam2

### Map

当使用 map() 函数返回泛型类型的时候也会发生类似的问题。下例中的方法签名 Tuple2<Integer, Integer> map(Integer value) 被擦除为 Tuple2 map(Integer value)。

例子：

**package** com.sq3;  
  
**import** jdk.nashorn.internal.codegen.types.Type;  
**import** org.apache.flink.api.common.typeinfo.Types;  
**import** org.apache.flink.api.java.tuple.Tuple2;  
**import** org.apache.flink.streaming.api.datastream.DataStream;  
**import** org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;  
  
**public class** lam3 {  
 **public static void** main(String[] args) **throws** Exception {  
 StreamExecutionEnvironment env = StreamExecutionEnvironment.*getExecutionEnvironment*();  
 DataStream<String> input = env.fromElements(**"a"**,**"b"**,**"c"**);  
 input.map(a-> Tuple2.*of*(a,1))  
 .returns(Types.*TUPLE*(Types.***STRING***,Types.***INT***))  
 .print();  
 env.execute(**"lam3"**);  
 }  
}