语法糖

Lambda表达式理解为简洁地表示可传递的匿名函数的一种方式：它没有名称，但它有参数列表、函数主体、返回类型，可能还有一个可以抛出的异常列表

标准的Lambda表达式

```
1. list.sort( (User u1 , User u2) -> {
2.     int i = u1.getCode().compareTo(u2.getCode());
3.     return i;
4. });
```

参数列表：`User u1 , User u2`
函数主体： 用大括号包起来的{}
返回类型：return i;

简写一（参数简写）

```
1. list.sort( (u1 , u2) -> {
2.     int i = u1.getCode().compareTo(u2.getCode());
3.     return i;
4. });
```

说明：因为这里list里面存储的就是 User类型，所以可以不用写参数类型

简写二（返回值简写）

```
1. list.sort( (u1 , u2) ->
2.     u1.getCode().compareTo(u2.getCode())
3. );
```

这里如果处理逻辑不复杂，一行就可以搞定，那么不需要用大括号包起来，这行代码如果有返回值就会自动返回，也不需要显示return

什么时候使用Lambda表达式

可以在函数式接口上使用Lambda表达式

函数式接口就是**只定义了一个抽象方法**的接口（一个接口可以有多个方法，但是只能有且只有一个抽象方法）

方法引用

```
1. button.setOnAction(event -> System.out.println(event));
```

通过方法引用可以写成

```
1. button.setOnAction(System.out::println);
```

假设想不区分大小写地对字符串进行排序，方法引用的写法为

```
1. Arrays.sort(strings, String::compareToIgnoreCase);
```

::操作符将方法名和对象或类的名字分隔开来。一下是主要的三种使用情况：

对象 :: 实例方法
类 :: 静态方法
类 :: 实例方法

前两种情况中，方法引用等同于提供方法参数的lambda表达式。比如：Math::pow等同于(x, y) -> Math.pow(x, y)。
第三种情况中，第一个参数会成为执行方法的对象。比如：String::compareToIgnoreCase等同于(x, y) -> x.compareToIgnoreCase(y)。

可以捕获方法引用中的this参数。例如：this::equals 等同于 x -> this.equals(x)

也可以使用super对象：

```
1. class Greeter {
2.     public void greet(){
3.         System.out.println("Hello world!");
4.     }
5. }
6.
7. class ConcurrentGreeter extends Greeter {
8.     public void greet(){
9.         Thread t = new Thread(super::greet);
10.         t.start();
11.     }
12. }
```

Lambda 和方法引用实战

List<Apple> inventory = new arrayList();

目标：推导出

```
1. inventory.sort(comparing(Apple::getWeight));
```

第一步：传递代码

```
1. public class AppleComparator implements Comparator<Apple> {
2.         public int compare(Apple a1, Apple a2){
3.         return a1.getWeight().compareTo(a2.getWeight());
4.     }
5. }
6. inventory.sort(new AppleComparator());
```

第二步：使用匿名内部类

```
1. inventory.sort(new Comparator<Apple>() {
2.         public int compare(Apple a1, Apple a2){
3.         return a1.getWeight().compareTo(a2.getWeight());
4.     }
5. });
```

第三步：使用lambda表达式

Comparator接口提供一个compare方法，传递两个参数，并且返回一个int值，那么用lambda表达式可以写成

```
1. inventory.sort((Apple a1, Apple a2)
2.     -> a1.getWeight().compareTo(a2.getWeight())
3. );
```

由于inventory中存储的就是Apple类型，所以可以简写为

```
4. inventory.sort((a1, a2)
5.     -> a1.getWeight().compareTo(a2.getWeight())
6. );
```

Comparator中提供了一个静态方法comparing()， 接收一个参数 Function ， 返回Comparator对象.

而Function 接收一个参数，返回一个参数

所以可以推导出下面代码

```
1. Comparator<Apple> c = Comparator.comparing((Apple a) ->
a.getWeight());
```

那么可以写的精简一点

```
1. import static java.util.Comparator.comparing;
2. inventory.sort(comparing((a) -> a.getWeight()));
```

第四步：使用方法引用

```
1. inventory.sort(comparing(Apple::getWeight));
```

复合 Lambda 表达式的有用方法

比较器复合用法

逆序： 按重量递减排序

inventory.sort(comparing(Apple::getWeight).reversed());

比较器链：按重量递减排序，两个苹果一样重时，进一步按国家排序

```
1. inventory.sort(comparing(Apple::getWeight)
2.      .reversed()
3.      .thenComparing(Apple::getCountry));
```

谓词复合

```
1. Predicate<Apple> redAndHeavyAppleOrGreen = redApple.and(a ->
a.getWeight() > 150)
2.                                                      .or(a ->
"green".equals(a.getColor()));
```

函数复合

简单示例

```
1. Function<Integer, Integer> f = x -> x + 1;
2. Function<Integer, Integer> g = x -> x * 2;
3. Function<Integer, Integer> h = f.andThen(g);
4. int result = h.apply(1);
```

流水线式（链式代码）

```
1. public class Letter{
2.     public static String addHeader(String text){
3.         return "From Raoul, Mario and Alan: " + text;
4.     }
5.     public static String addFooter(String text){
6.         return text + " Kind regards";
7.     }
8.     public static String checkSpelling(String text){
9.         return text.replaceAll("labda", "lambda");
```

```
    }
}
```

```
Function<String, String> addHeader = Letter::addHeader;
Function<String, String> transformationPipeline =
addHeader.andThen(Letter::checkSpelling)
.andThen(Letter::addFooter);
```