java中（几乎）所有一切都是对象，java中没有函数类型，所以函数也被表达为对象，也就是实现了特定接口的类的实现，lambda表达式提供了一种便捷的语法来创建这样的实例。

比如我们比较两个字符串的长度

str1.**length**() - str2.**length**()

但是java是强类型的语言，必须同时指定类型才可以

(String str1, String str2) -> str1.**length**() - str2.**length**()

这样就是一个简单的lambda表达式了，不需要指定返回值的类型，编译器会从lambda表达式推断出返回的类型，并且会检查返回类型是否与期望的类型符合。比如上面的lambda表达式，可以被用在期望结果为int型兼容的地方，比如int、long、Integer等。

我们通常比较两个字符串的长度需要定义一个方法，比如：

public int **compare**(String str1, String str2) {

return str1.**length**() - str2.**length**();

}

如果lambda表达式的表达体执行一个无法用一个表达式表示的计算，那么需要用编写方法的方式来编写，例如：

(String first, String second) -> {

int diff = first.**length**() - second.**length**();

if(diff < 0) return -1;

else if(deff > 0) return 1;

else return 0;

}

如果lambda表达式没有参数，使用空的括号就可以了

() -> { for(int i = 0; i < 10; i++) out.**println**("hello"); }

如果lambda表达式的参数类型可以被推到出来，括号中可以省略类型

Comparator<String> comp = (first, second) -> first.**length**() - second.**length**();

# 函数式接口

当一个接口只有一个抽象方法（其余的都是static或者default）的时候，比如继承Comparator接口只要重写compare方法，就可以提供一个lambda表达式来代替。

我们想利用Arrays.sort根据字符串长度进行排序时，需要提供第二个参数是一个实现了Comparator接口的实例，但是也可以用lambda表达式代替。

Comparator<String> comp = (first, second) -> first.**length**() - second.**length**();

Arrays.**sort**(names, comp);

//或者是

Arrays.**sort**(names, (first, second) -> first.**length**() - second.**length**());

Arrays.sort方法的第二个参数变量接受一个实现了Comparator接口的类的实例，调用该对象的compare方法会执行lambda表达式中的代码，所以这也就是为什么接口只有一个抽象方法的时候可以用lambda表达式代替。

# 方法引用

假设想不区分大小写对字符串进行排序，可以这样写：

Arrays.**sort**(names, (x, y) -> x.**compareIgnoreCase**(y))

也可以用下面的方式

Arrays.**sort**(names, String::compareIgnoreCase)

String::compareIgnoreCase是方法引用，等同于上面的lambda表达式。

ArrayList的removeIf方法可以将数组列表中为null的删除，removeIf的参数是Predicate类型，这个接口是函数式接口

list.**removeIf**(e -> e == null)

Objects有一个isNull方法用来判断对象是否为null，所以上面的也可以写成这样：

list.**remveIf**(Objects::isNull)

ArrayList的forEach会在所有元素执行一个函数，比如：

list.**forEach**(x -> System.out.**println**(x))

还可以写成：

list.**forEach**(System.out::println)

上面的三个例子，分别是：

1. 类::实例方法
2. 类::静态方法
3. 对象::实例方法

# 使用lambda表达式

前面的都是如何把lambda表达式传递给期望函数式接口的方法，下面如何写试用lambda表达式的方法。

## 延迟执行

使用lambda就在于要延迟执行，如果只是立即执行，那也就无需把代码封装进lambda了，直接调用就可以了。延迟执行的原因，比如：

1. 在另外一个线程中运行代码；
2. 在算法的恰当时刻运行代码；
3. 在某些情况发生是运行代码；
4. 只有在需要的时候才运行代码。

比如现在需要重复执行一个行为

**repeat**(10, () -> System.out.**println**("hello"))

要接收lambda表达式，我们需要选择函数式接口，比如上面的例子可以使用Runnable：

public static void **repeat**(int n, Runnable action){

for(int i = 0; i < n; i++)

action.**run**();

}

当action.run方法执行的时候，lambda表达式体被执行。

如果想要让action执行的时候知道是在第几次迭代，那么action应当可以收到一个int型的参数，所以我们需要一个函数式接口，它有个参数类型为int，返回值为void的方法，java提供了一些常用的函数式接口，这里可以使用IntConsumer。

**repeat**(10, (i) -> System.out.**println**("hello" + i))

public static void **repeat**(int n, IntConsumer action){

for(int i = 0; i < n; i++)

action.**accept**(i);

}