**Lambda表达式**

1. Lambda表达式

从JDK1.8开始，为了简化使用者的代码开发，专门提供有Lambda表达式的支持，利用此操作可以实现函数式的编程，对于函数式的编程比较著名的式haskell ，Scala，利用函数的编程我们可以避免掉面向对象中的繁琐处理问题。

面向对象在其长期发展的过程中，有一部分的反对者，这些反对者一致认为面向对象的开发过程过于繁琐。

以一个最简单的程序为例：

范例：

|  |
| --- |
| interface IMessage {  public void send (String str ) ;  }  public class JavaDemo {  public static void main(String[] args) {  IMessage msg = new IMessage(){  public void send (String str ){  System.out.println("消息发送："+ str);  }  };  msg.send("我很好");  }  } |

**在这样的一个简单程序中，我们的核心语句只有一行语句！“ System.out.println("消息发送："+ str);”。**

而为了这么一行语句我们依然需要使用面向对象的设计流程，于是这些问题随着技术的发展不断突出，java不得不提供Lambda表达式。

下面我们使用lamabda表达式：

范例：（lambda表达式实现相同功能。）

|  |
| --- |
| interface IMessage {  public void send (String str ) ;  }  public class JavaDemo {  public static void main(String[] args) {  IMessage msg = ( str ) ->{  System.out.println("发送消息："+ str);  } ;  msg.send("我很好");  }  } |

但是我们需要注意:

Lambda表达式如果想要使用，那么必须有一个很重要的实现要求：SAM （Single Abstract Method）只有一个抽象方法，以之前的IMessage 接口为例，里面只有一个send() 方法，除此之外没有任何其它方法定义，所以这样的接口就称为函数式接口，而只有函数式接口才能使用Lamda表达式；

而为了明确的表达他是一个函数式的接口我们往往会在使用@FunctionalInterface的注解来表示；

|  |
| --- |
| @FunctionalInterface  interface IMessage {  public void send (String str ) ;  }  public class JavaDemo {  public static void main(String[] args) {  IMessage msg = ( str ) ->{  System.out.println("发送消息："+ str);  } ;  msg.send("我很好");  }  } |

而一旦使用了@FunctionalInterface注解，该接口中就只能拥有一个方法。

此处我们可以联想到，如果接口中有些公共方法怎么办呢？

此时，default static 关键字就起作用了。

我们可以使用公共方法，使用default 和 static 来修饰方法。

从而达到目的；

对于Lambda表达式而言，提供有如下几种格式:

·方法没有参数: () ->{} ;

·方法有参数: (参数，参数)->{} ;

·如果只有一行语句: (参数，参数)->语句 ；

1. 方法引用
   1. 引用数据类型最大的特点就是可以进行内存的指向处理，但是在传统开发之中一直使用的只是对象的引用操作，而JDK1.8之后我们也提供有方法的引用，即：不同的方法名称可以描述同一个方法。如果要进行方法的引用在Java之中提供有以下的四种模式:
      1. 引用静态方法: 类名称::static 方法名称；
      2. 引用某个实例对象的方法: 实例化对象:: 普通方法;
      3. 引用特点类型的方法: 特点类:: 普通方法 ；
      4. 引用构造方法 : 类名称:: new .

范例:引用静态方法

·在String类里面提供有String.valueOf()方法，这个方法他就属于静态方法。

-方法定义: public static String valueOf​(int i)

|  |
| --- |
| @FunctionalInterface  // p描述的是参数，r描述的是返回值  interface IFunction<P,R> {  public R change (P p) ;  }  public class JavaDemo2{  public static void main(String[] args) {  **IFunction<Integer , String > fun = String :: valueOf ;**  String str = fun.change(100) ;  System.out.println(str.length());  }  } |

利用方法引用这一概念可以为一个放=方法定义多个名字，但是必须是函数式接口。

范例: 引用实例化对象中的方法。

·String类中有一个转换大写的方法。

-这个方法是必须在有实例化对象提供的情况下才可以调用；

|  |
| --- |
| @FunctionalInterface  // p描述的是参数，r描述的是返回值  interface IFunction<P> {  public P upper () ;  }  public class JavaDemo2{  public static void main(String[] args) {  IFunction< String > fun = "www.baidu.com":: toUpperCase ;  System.out.println(fun.upper());  }  } |