1. Lambda表达式

以函数式编程思想来解决问题，即只考虑做什么，而不考虑以什么形式来做，忽略复杂的语法，用最简单的方式解决问题。

1. 格式

(形式参数) -> {代码块}

形式参数：如果有多个参数，参数之间用逗号隔开；如果没有参数，留空即可

-> ：由英文中画线和大于符号组成，固定写法。代表指向动作

代码块：是我们具体要做的事情，也就是以前我们写的方法体内容

1. Lambda表达式使用的前提
2. 有一个接口
3. 接口中有且只有一个抽象方法
4. 使用步骤
5. 定义一个接口，里面定义一个抽象方法
6. 定义一个测试类，在测试类中定义一个private方法，用于内部调用。
7. 主方法中调用private方法，使用Lambda表达式

使用Lambda表达式是最简洁的，也可以使用匿名内部类，或者用接口的实现类。

// 接口

public interface Eatable {

  void eat();

}

//实现类

public class EatableImpl implements Eatable {

  @Override

  public void eat() {

    System.out.println("一天一苹果，医生远离我");

 }

}

//测试类

public class EatableDemo {

  public static void main(String[] args) {

    //在主方法中调用useEatable方法

Eatable e = new EatableImpl();

useEatable(e);

    //匿名内部类

    useEatable(new Eatable() {

      @Override

      public void eat() {

        System.out.println("一天一苹果，医生远离我");

     }

   });

    //Lambda表达式

    useEatable(() -> {

      System.out.println("一天一苹果，医生远离我");

   });

 }

  private static void useEatable(Eatable e) {

    e.eat();

 }

}

1. Lambda表达式的省略规则
2. 参数类型可省略，且必须同时省略
3. 若只有一个参数，则参数的括号可省略
4. 若代码块只有一句，可省略大括号，分号和return
5. Lambda表达式与匿名内部类的区别
6. 匿名内部类可以用于接口，抽象类或者具体类;而Lambda表达式只能用于接口。
7. Lambda表达式只能在接口中仅有一个抽象方法时使用，而匿名内部类没有这种限制。
8. 匿名内部类编译后产生一个单独的.class字节码文件，而Lambda表达式的字节码文件会在运行时动态生成。
9. 接口的组成
10. 接口的组成
11. 常量 public static final
12. 抽象方法 public abstract
13. 默认方法（Java 8）default
14. 静态方法（Java 8）static
15. 私有方法（Java 9）private
16. 默认方法
17. 格式

Public default 返回值类型 方法名(参数列表) { }

1. 注意事项

默认方法是接口的默认存在的方法，可以重写，但不是强制的，重写时需要去掉default关键字。

1. 静态方法
2. 格式

public static 返回值类型 方法名(参数列表) { }

1. 注意事项

静态方法只能通过接口名调用，不能通过实现类名或者对象名调用。

1. 私有方法

当接口中的默认方法或者静态方法中有相同的代码时，可以将该代码块抽出来作为接口的私有方法只供内部使用。

1. 格式

private 返回值类型 方法名(参数列表) { }

private static 返回值类型 方法名(参数列表) { }

1. 注意事项

默认方法可以调用私有的静态方法和非静态方法

静态方法只能调用私有的静态方法

1. 方法引用

直接引用已存在的方法，不需要自己重新写一个方法。

1. 方法引用符
2. 格式

“ ：： ”，两个冒号，表示引用运算符。

1. 注意事项

调用方法引用就是调用已存在的方法，该方法的参数会根据上下文被推导出来。

1. 引用类方法

引用类的静态方法，可以替换Lambda表达式，它的形参全部传递给静态方法。

1. 格式：

类名：：静态方法

1. 示例

public interface Converter {

  int convert(String s);

}

public class ConverterDemo {

  public static void main(String[] args) {

//Lambda写法

    useConverter(s -> Integer.parseInt(s));

    //引用类方法

    useConverter(Integer::parseInt);

 }

  private static void useConverter(Converter c) {

    int number = c.convert("666");

    System.out.println(number);

 }

}

1. 引用对象的实例方法

引用类中的成员方法

1. 格式：  
   对象：：成员方法
2. 示例

public class PrintString {

  //把字符串参数变成大写的数据，然后在控制台输出

  public void printUpper(String s) {

    String result = s.toUpperCase();

    System.out.println(result);

 }

}

public interface Printer {

  void printUpperCase(String s);

}

public class PrinterDemo {

  public static void main(String[] args) {

//Lambda简化写法

    usePrinter(s -> System.out.println(s.toUpperCase()));

    //引用对象的实例方法

    PrintString ps = new PrintString();

    usePrinter(ps::printUpper);

 }

  private static void usePrinter(Printer p) {

    p.printUpperCase("HelloWorld");

 }

}

1. 引用类的实例方法

引用类的成员方法，与对象的实例方法不同的是，没有创建对象，直接用类名调用。且形参中第一个参数作为调用者，后面的参数才是真正传给该方法的参数。

1. 格式

类名：：成员方法

1. 示例

public interface MyString {

  String mySubString(String s,int x,int y);

}

public class MyStringDemo {

  public static void main(String[] args) {

//Lambda简化写法

    useMyString((s,x,y) -> s.substring(x,y));

    //引用类的实例方法

    useMyString(String::substring);

 }

  private static void useMyString(MyString my) {

    String s = my.mySubString("HelloWorld", 2, 5);

    System.out.println(s);

 }

}

1. 引用构造器

引用类的构造方法

1. 格式

类名：：new

2， 示例

Student::new