QST 青软实训

第八章 JAVA 8新特性

本章重点

- 掌握接口默认方法和静态方法的使用
- 掌握Lambda表达式的规范和使用
- 掌握函数式接口的使用
- 了解::方法引用
- 了解JAVA 8新增类库

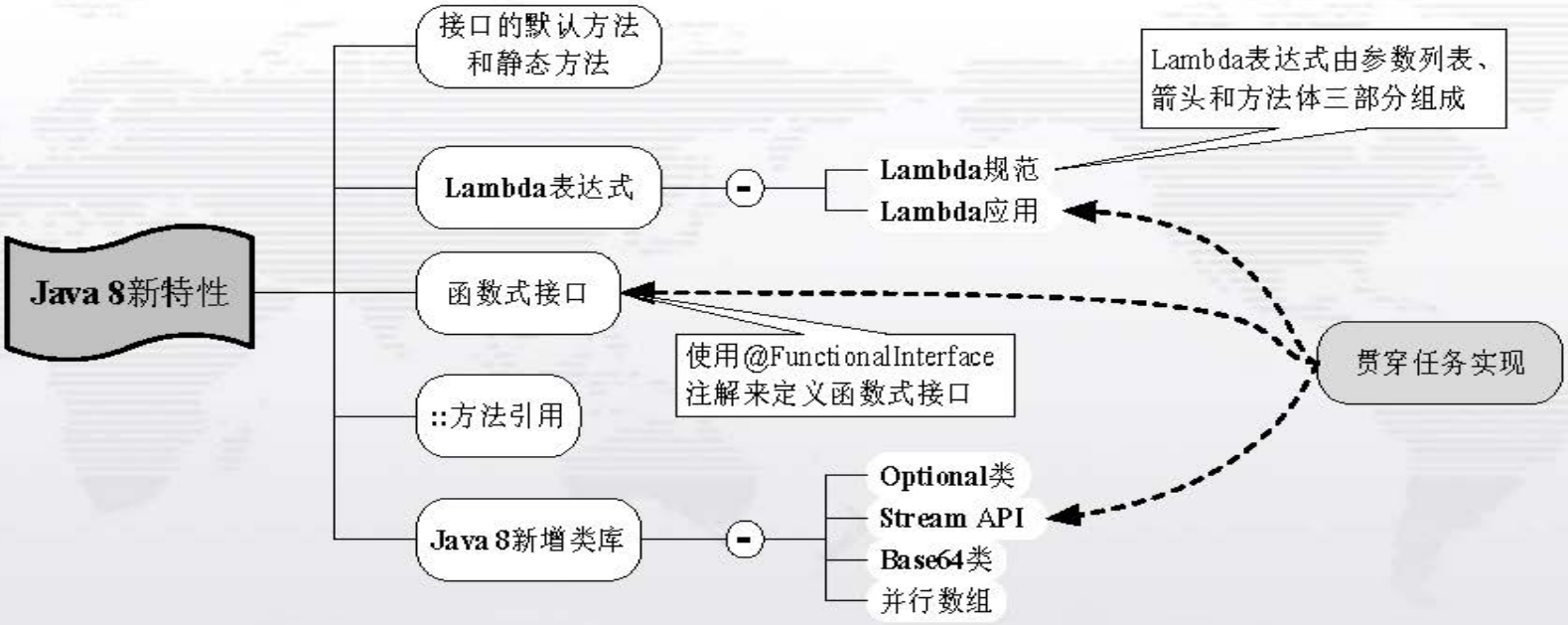
任务驱动

- **本章任务是使用Lambda迭代升级“Q-DMS数据挖掘”系统中的部分代码，具体要求如下：**

【任务8-1】 使用Lambda表达式迭代升级主窗口中“帮助”菜单的事件处理。

【任务8-2】 使用Lambda表达式实现查找指定的匹配信息并显示。

学习路线


Java 8新特性
接口的默认方法
和静态方法
Lambda表达式
Lambda规范
Lambda应用
Lambda表达式由参数列表、
箭头和方法体三部分组成
函数式接口
使用@FunctionalInterface
注解来定义函数式接口
::方法引用
Java 8新增类库
Optional类
Stream API
Base64类
并行数组
贯穿任务实现

本章目标

知识点	听	看	抄	改	写
接口默认方法和静态方法	★	★	★	★	
Lambda表达式	★	★	★	★	
函数式接口	★	★	★	★	★
方法引用	★	★	★		
Java 8新增类库	★	★	★		

8.1 接口的默认方法和静态方法

从Java 8开始允许在接口中定义默认方法和静态方法

- 默认方法又称为扩展方法，需要在方法前使用default关键字进行修饰
- 静态方法就是类方法，需要在方法前使用static关键字进行修饰

注意

接口中定义的普通方法不管是否使用public abstract进行修饰，其默认总是使用public abstract来修饰，即接口中的普通方法都是抽象方法，不能有方法的实现（方法体）；而接口的默认方法和静态方法都必须有方法的实现（方法体）。

讲师演示讲解

【代码8-1】IDefaultStatic.java

- 默认方法虽然有方法体，但不能通过接口直接访问，必须通过接口实现类的实例进行访问

【语法】访问默认方法

```
对象名.默认方法名()
```

- 静态方法则可以直接通过接口名进行访问，也可以通过接口实现类的实例进行访问

【语法】访问默认方法

```
接口名.静态方法名()
```

讲师演示讲解

【代码8-2】IDefaultStaticDemo.java

8.2.1 Lambda规范

- Lambda表达式支持将代码作为方法参数，允许使用更加简洁的代码来创建只有一个抽象方法的接口实例
- 【语法】普通方法的定义

```
返回类型 方法名(参数列表) {
        方法体
}
```

- 【语法】 Lambda表达式（匿名方法）

```
(参数列表) -> { 方法体 }
```

  - 参数列表中的参数都是匿名方法的形参，即输入参数
  - ->箭头是Lambda运算符
  - 方法体可以是单一的表达式或多条语句组成的语句块

参数列表允许省略形参的类型，即这些参数可以是明确类型和推断类型；当参数是推断类型时，参数的数据类型由JVM根据上下文自动推断出来。

如果只有一条语句，允许省略方法体的{}；如果只有一条return语句，可以省略return关键字。
Lambda表达式需要返回值，如果方法体仅有一条省略了return关键字的语句，Lambda表达式会自动返回该条语句的结果值。

- 【示例】多参数、多语句的Lambda表达式

```
(int x, int y) -> {
        System.out.println(x);
        System.out.println(y);
        return x+y;
}
```

- 【示例】省略参数类型的Lambda表达式

```
(int x, int y) -> {
        System.out.println(x);
        System.out.println(y);
        return x+y;
}
```

- 【示例】省略{}和return关键字的Lambda表达式

```
(x, y) -> x + y
```

- 【语法】只有一个参数的Lambda表达式

```
参数名 -> { 方法体 }
```

- 【示例】省略()的Lambda表达式

```
x -> {
        System.out.println(x);
        return ++x;
}
```

- 【语法】只有一个参数和一条语句的Lambda表达式

```
参数名 -> 表达式
```

- 【示例】省略()、{}和return关键字的Lambda表达式

```
x -> ++x
```

- 【语法】没有参数的Lambda表达式

```
() -> { 方法体 }
```

- 【示例】没有参数的Lambda表达式

```
//没有参数,返回值为5
() -> 5
//没有参数，只打印信息
() -> System.out.println("QST欢迎您!")
```

8.2.2 Lambda应用

- 使用Lambda表达式输出集合内容

讲师演示讲解

【代码8-3】LambdaDemo.java

注意

Java 8提供双冒号操作符::，该操作符用于方法引用。

- 使用Lambda表达式实现排序

讲师演示讲解

【代码8-4】LambdaDemo2.java

- 使用Lambda表达式实现监听器

讲师演示讲解

【代码8-5】LambdaDemo3.java

8.3 函数式接口

- 函数式接口本质上是一个仅有一个抽象方法的普通接口
- 函数式接口能够被隐式地转换为Lambda表达式
- Java 8增加了一个注解@FunctionalInterface来定义函数式接口

为了防止别人在接口中添加方法导致其不再是函数式接口。

【语法】

```
@FunctionalInterface
public interface 接口名 {
        //只有一个抽象方法
}
```

注意

@FunctionalInterface注解已经在第7章基本注解中介绍过，使用该注解修饰的接口必须是函数式接口，该接口中只能声明一个抽象方法，如果声明多个抽象方法则会报错。但是默认方法和静态方法不属于抽象方法，因此在函数式接口中也可以定义默认方法和静态方法。

讲师演示讲解

【代码8-6】FIConverter.java

【代码8-7】FunctionalInterfaceDemo.java

注意

Lambda表达式只能为函数式接口创建对象，即Lambda表达式只能实现具有一个抽象方法的接口，且该接口必须是由@FunctionalInterface注解修饰的函数式接口。

8.4 ::方法引用

- 在Java 8中可以使用双冒号操作符::来简化Lambda表达式，其语法

```
容器::方法名
```

适用 引用的方法：

- ::方法引用静态方法

```
类名::静态方法名
```

- ::方法引用实例方法

```
对象::实例方法名
```

- ::方法引用构造方法

```
类名::new
```

- 【示例】::静态方法引用

```
//String.valueOf()静态方法
String::valueOf
//Math.max()静态方法
Math::max
```

讲师演示讲解

【代码8-8】MethodReferenceDemo.java

【代码8-9】ConstructorMethodReferenceDemo.java

注意

无论是方法引用还是Lambda表达式，其最终原理和所要实现的就是当某一个类中，或者接口中的某一方法，其入参为一个接口类型时，使用方法引用或者Lambda表达式可以快速而简洁的实现这个接口，而不必繁琐的创建一个这个接口的对象或者直接实现。

8.5.1 Optional类

Optional类是为了解决空指针异常问题。

- Optional类实际上是个容器，可以保存类型T的值，或者仅保存null
- Optional类常用的方法

方法	功能描述
public T orElse(T other)	通过工厂方法创建Optional类。注意创建对象时传入的参数不能为null，如果传入的参数为null，则抛出NullPointerException
public static <T> Optional<T> ofNullable(T value)	为指定的值创建一个Optional，如果指定的值为null，则返回一个空的Optional；
public boolean isPresent()	如果值存在返回true，否则返回false
public T get()	如果Optional有值则将其返回，否则抛出NoSuchElementException异常
public void ifPresent(Consumer<? super T> consumer)	如果Optional实例有值则为其调用consumer，否则不做处理。Consumer类包含一个抽象方法，该抽象方法可以对传入的值进行处理，但没有返回值

方法	功能描述
public T orElse(T other)	如果有值则将其返回，否则返回指定的other值
public T orElseGet(Supplier<? extends T> other)	orElseGet()与orElse()方法类似，区别在于得到的默认值，orElse()方法可将传入的字符串作为默认值，而orElseGet()方法可以接受Supplier接口的实现来生成默认值
public <X extends Throwable> T orElseThrow(Supplier<? extends X> exceptionSupplier) throws X extends Throwable	如果有值则将其返回，否则抛出Supplier接口创建的异常
public <U> Optional<U> map(Function<? super T,? extends U> mapper)	如果有值，则对其执行该方法得到返回值；如果返回值不为null，则创建包含返回值的Optional，否则返回空Optional
public <U> Optional<U> flatMap(Function<? super T,Optional<U>> mapper)	如果有值，为其执行该方法返回Optional类型的返回值，否则返回空Optional。flatMap()与map()方法类似，区别在于flatMap()中的mapper返回值必须是Optional，调用结束时，flatMap不会对结果用Optional封装
public Optional<T> filter(Predicate<? super T> pr	通过传入限定条件对Optional实例的值进行过滤

讲师演示讲解

【代码8-10】OptionalDemo.java

8.5.2 Stream API

- java.util.stream包中提供了一些Stream API， Stream API将真正的函数式编程风格引入到Java中

可以使得代码更加干净，简洁。极大简化了集合框架的处理，提高了效率和生产力。

讲师演示讲解

【代码8- 11】StreamDemo.java

8.5.3 Base64类

- java.util.Base64工具类提供了一套静态方法获取Basic、URL和MIME三种Base64编码和解码器
- 其常用的方法

方法	功能描述
public static Base64.Encoder getEncoder()	获取基于Basic类型的Base64编码器
public static Base64.Encoder getUrlEncoder()	获取基于URL类型的Base64编码器
public static Base64.Encoder getMimeEncoder()	获取基于MIME类型的Base64编码器
public static Base64.Decoder getDecoder()	获取基于Basic类型的Base64解码器
public static Base64.Decoder getUrlDecoder()	获取基于URL类型的Base64解码器
public static Base64.Decoder getMimeDecoder()	获取基于MIME类型的Base64解码器

讲师演示讲解

【代码8-12】Base64Demo.java

8.5.4 并行数组

- Java 8中支持并行数组操作，并增加了大量使用Lambda表达式的新方法用于对数组进行排序、过滤和分组等操作
- 【示例】使用parallelSort()方法排序

```
Arrays.parallelSort(numbers);
```

- 【示例】使用groupingBy()方法分组

```
Map<Boolean, List<Integer>> groupByOdd = numbers.parallelStream()
                    .collect(Collectors.groupingBy(x -> x % 2 == 0));
```

- 【示例】使用filter()方法过滤

```
Integer[] evens = numbers.parallelStream().filter(x -> x % 2 == 0).toArray();
```

讲师演示讲解

【代码8-13】 ParallelArraysDemo.java

8.6 贯穿任务实现

- 【任务8-1】使用Lambda表达式迭代升级主窗口中“帮助”菜单的事件处理。
  - *MainFrame.java*
- 【任务8-2】使用Lambda表达式实现查找指定的匹配信息并显示。
  - *IDataAnalyse.java*
  - *LogRecService.java*
  - *TransportService.java*
  - *LambdaSearchDemo.java*

本章总结

- 默认方法又称为扩展方法，需要在方法前使用default关键字进行修饰
- 静态方法就是类方法，需要在方法前使用static关键字进行修饰
- Lambda表达式是Java 8更新的重要新特性
- Lambda表达式由参数列表、箭头和方法体三部分组成
- 函数式接口使用@FunctionalInterface注解来定义
- ::方法引用可以更加简化Lambda表达式
- Optional类通过使用检查空值的方式来防止代码污染
- Stream API将真正的函数式编程风格引入到Java中
- Base64工具类提供一套静态方法获取Basic、URL和MIME三种Base64编码和解码器

QST 青软实训

信·未来