

Java8

superliu213

Published
with GitBook



Table of Contents

Introduction	0
Java8 新特性	1
Lambda 概述	2
Stream 接口	3
过滤器案例	4
DoubleStream 接口	5
LocalDate 类	6
LocalTime 类	7
LocalDateTime 类	8
DateTimeFormatter 类	9
ZonedDateTime 类	10

Java 8 新特性

主要是介绍了Java8 的新特性，并展示了一些简单案例，方便快速地了解 and 掌握其新特性。

Java8 新特性

发布

1. oracle公司与2014年3月27日正式推出Java8
2. Java SE 8
3. Java ME 8
4. Java Embedded8

Java SE8的重要性能包括：

1. Lambda
函数式语法规则，在C#中早有应用
2. Nashorn JavaScript引擎

```
import javax.script.ScriptEngine;
import javax.script.ScriptEngineManager;

public class Hello {
    public static void main(String... args) throws Throwable {
        ScriptEngineManager engineManager = new ScriptEngineManager();
        ScriptEngine engine = engineManager.getEngineByName("nashorn");
        engine.eval("function sum(a, b) { return a + b; }");
        System.out.println(engine.eval("sum(1, 2);"));
    }
}
```

[Oracle Nashorn: A Next-Generation JavaScript Engine for the JVM](#)

3. 新的日期与时间API
4. 一套简洁的配置文件
5. 从JVM中去除了“永久代(permanent generation)”
提高了12%到14%的性能
6. 增强的注解功能
可以在编译期间就检查出来，以前只能在运行期才发现的例如空指针等异常错

误

Java SE Embedded8

1. Java SE Embedded8 为嵌入式及物联网设备提供了一个开发平台，具备Java SE8 的灵活性、可移植性和功能
2. Java SE Embedded8 使开发人员能够运用Java SE8 为嵌入式设备建立更小的平台

Java ME8

专用于诸如机顶盒等移动设备的开发

1. 与Java SE8一致的Java语言和API
2. 支持最新的Web协议
3. 全面的应用模型
4. 先进的安全功能
5. 用于电源管理及与多种标准外部设备交互的标准API

Lambda 概述

Lambda概述

lambda表示数学符号“ λ ”，计算机领域中 λ 代表“ λ 演算”，表达了计算机中最基本的概念：“调用”和“置换”

为什么使用lambda

- 一、Java是面向对象的语言，不能象函数式语言那样嵌套定义方法
- 二、Java的匿名内部类只能存在于创建它的线程中，不能运行在多线程中，无法充分利用多核的硬件优势
- 三、匿名内部类的缺点还有：
 1. 语法相对复杂
 2. 在调用内部类的上下文中，指引和this的指代容易混淆
 3. 类加载和实例创建语法不可避免
 4. 不能引用外部的非final对象
 5. 不能抽象化控制流程在4和5上只是部分进行了优化，没有完全解决，例如虽然可以引用外部的非final对象，但是这个对象还是不可以进行修改，例如自增操作

Lambda语法

Lambda的语法包括三部分

1. 参数列表
2. 箭头符号" \rightarrow "
3. 代码块。

案例

1. 用lambda简化Runnable接口的实现方式

```
public class Test_Lambda01{
    public static void main(String[] args){
        //传统方式定义匿名内部类
        new Runnable(){
            @Override
            public void run(){
                System.out.println("匿名内部类实现Runnable接口");
            }
        }.run();

        //Lambda方式
        int i = 1;
        Runnable r = ()->{
            System.out.println("用Lambda类实现Runnable接口");
            System.out.println("i="+i);
        };
        r.run();
    }
}
```

2. lambda实现自定义接口，模拟登陆操作

```
public class Test_Lambda02{
    public static void main(String[] args){
        new Action(){
            @Override
            public void execute(String content){
                System.out.println(content);
            }
        }.execute("jdk1.8之前的匿名内部类实现方式，执行登录操作");
        Action login = (String content)->{
            System.out.println(content);
        };
        login.execute("jdk1.8的Lamda语法实现登录操作");
    }

    static interface Action{
        void execute(String content);
    }
}
```


Stream 接口

什么是Stream

1. Stream在Java8中被定义为泛型接口
2. Stream接口代表数据流
不同于IO流
3. Stream不是一个数据结构，不直接存储数据
4. Stream通过管道操作数据
5. 创建Stream接口实现类对象：
stream()：创建一个Stream接口实现类的对象
例如：

```
//people是一个Arraylist集合  
Stream<Person> stream=people.stream();
```

什么是管道

- 一、管道：代表一个操作序列
- 二、管道包含以下组件：
 1. 数据集：可能是集合、数组等
 2. 零个或多个中间业务，如过滤器
 3. 一个终端操作，如的forEach

什么是过滤器

1. 过滤器：用给定的条件在源数据基础上过滤出新的数据，并返回一个Stream对象
2. 过滤器包含匹配的谓词 例如：判断p对象是否为男性的lambda表达式：

```
Stream<Person> stream=people.stream();  
stream=stream.filter(p->p.getGender()=='男');
```

案例

创建一个元素为Person类的集合：people

使用Stream和forEach显示该集合所有元素

```
public class Person {  
    public static enum Sex {  
        Male, Female;  
    }  
  
    private String name;  
    private Sex gender;  
    private int age;  
    private double height;  
  
    public Person(String name, Sex gender, int age, double height) {  
        this.name = name;  
        this.gender = gender;  
        this.age = age;  
        this.height = height;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public Sex getGender() {  
        return gender;  
    }  
  
    public void setGender(Sex gender) {  
        this.gender = gender;  
    }  
  
    public int getAge() {  
        return age;  
    }  
}
```

```
public void setAge(int age) {
    this.age = age;
}

public double getHeight() {
    return height;
}


public void setHeight(double height) {
    this.height = height;
}

@Override
public String toString() {
    return "Person{" +
        "name='" + name + '\'' +
        ", gender=" + gender +
        ", age=" + age +
        ", height=" + height +
        '}';
}
}

public class Test_collection_stream {
    public static void main(String[] args) {
        List<Person> people = createPeople();
        Stream<Person> stream = people.stream();
        stream.forEach(
            person -> System.out.println(person.toString())
        );
    }

    static List<Person> createPeople(){
        List<Person> pepole = new ArrayList<>();
        Person p = new Person("张飞", Person.Sex.Male, 33, 1.99);
        pepole.add(p);
        p = new Person("王菲", Person.Sex.Female, 32, 1.71);
        pepole.add(p);
        p = new Person("刘亦菲", Person.Sex.Female, 31, 1.69);
        pepole.add(p);
    }
}
```

```
        p = new Person("马飞", Person.Sex.Male, 33, 1.89);  
        pepole.add(p);  
        return pepole;  
    }  
}
```



案例

创建一个元素为Person类的集合：people

使用Stream、filter和forEach显示该集合中所有女性

```
public class Test_collection_filter {  
    public static void main(String[] args) {  
        List<Person> people = createPeople();  
        people.stream()  
            .filter(person -> person.getGender() == Person.Sex.FEMALE)  
            .forEach(person -> System.out.println(person.toString()));  
    }  
  
    static List<Person> createPeople(){  
        List<Person> pepole = new ArrayList<>();  
        Person p = new Person("张飞", Person.Sex.Male, 33, 1.99);  
        pepole.add(p);  
        p = new Person("王菲", Person.Sex.Female, 32, 1.71);  
        pepole.add(p);  
        p = new Person("刘亦菲", Person.Sex.Female, 31, 1.69);  
        pepole.add(p);  
        p = new Person("马飞", Person.Sex.Male, 33, 1.89);  
        pepole.add(p);  
        return pepole;  
    }  
}
```

DoubleStream 接口

- 一、DoubleStream接口表示元素类型是double的数据源
- 二、DoubleStream接口的常用方法：
 1. `max().getAsDouble()`：获取流中数据集的最大值
 2. `stream.min().getAsDouble()`：获取流中数据集的最小值
 3. `stream.average()`：获取流中数据集的平均值
- 三、获取DoubleStream 集合变量`mapToDouble()`

案例

统计people集合中姓名中包含“菲”字的平均身高

```
public class Test_collection_doubleStream {
    public static void main(String[] args) {
        List<Person> people = createPeople();
        double avgHeight = people.stream()
            .filter(person -> person.getName().indexOf("菲") >= 0)
            .mapToDouble(person -> person.getHeight())
            .average()
            .getAsDouble();
        System.out.println("包含菲字的所有人的身高：" + avgHeight + "米");
    }

    static List<Person> createPeople() {
        List<Person> pepole = new ArrayList<>();
        Person p = new Person("张飞", Person.Sex.Male, 33, 1.99);
        pepole.add(p);
        p = new Person("王菲", Person.Sex.Female, 32, 1.71);
        pepole.add(p);
        p = new Person("刘亦菲", Person.Sex.Female, 31, 1.69);
        pepole.add(p);
        p = new Person("马飞", Person.Sex.Male, 33, 1.89);
        pepole.add(p);
        return pepole;
    }
}
```

LocalDate 类

LocalDate类使用ISO日历表示年、月、日

LocalDate类的常用方法：

1. `LocalDate.now()`：获取系统当前日期
2. `LocalDate.of(int year,int month,int dayOfMonth)`
按指定日期创建`LocalDate`对象。
3. `getYear()`：返回日期中的年份。
4. `getMonth()`：返回日期中的月份。
5. `getDayOfMonth()`：返回月份中的日。

案例

用`LocalDate`获取当前日期

```
public class Test_LocalDate{  
    public static void main(String[] args){  
        LocalDate date = LocalDate.now();  
        System.out.println(date.getYear()+"年");  
        System.out.println(date.getMonthValue()+"月");  
        System.out.println(date.getDayOfMonth()+"日");  
        System.out.println(date.toString());  
    }  
}
```


LocalTime 类

LocalTime类用于表示一天中的时间

LocalTime类的常用方法：

1. LocalTime.now()：获取系统当前时间
2. LocalTime.of(int hour,int minute,int second)
按指定时间创建LocalTime对象
3. getHour()：返回小时
4. getMinute()：返回分钟
5. getSecond()：返回秒

案例

用LocalTime获取当前时间

```
public class Test_LocalTime{  
    public static void main(String[] args){  
        LocalTime time = LocalTime.now();  
        System.out.println(time.getHour()+"时");  
        System.out.println(time.getMinute()+"分");  
        System.out.println(time.getSecond()+"秒");  
        System.out.println(time.toString());  
    }  
}
```

LocalDateTime 类

LocalDateTime类用于表示日期和时间

LocalDateTime类的常用方法：

1. `LocalDateTime.now()`：获取系统当前时间
2. `LocalDateTime.of(int year,int month,int dayOfMonth, int hour,int minute,int second)`
按指定日期和时间创建`LocalDateTime`对象
3. `getYear()`：返回日期中的年份
4. `getMonth()`：返回日期中的月份
5. `getDayOfMonth()`：返回月份中的日
6. `getHour()`：返回小时
7. `getMinute()`：返回分钟
8. `getSecond()`：返回秒

案例

用`LocalDateTime`获取当前日期和时间

```
public class Test_LocalDateTime{
    public static void main(String[] args){
        LocalDateTime dateTime = LocalDateTime.now();
        System.out.println(dateTime.getYear()+"年");
        System.out.println(dateTime.getMonthValue()+"月");
        System.out.println(dateTime.getDayOfMonth()+"日");
        System.out.println(dateTime.getHour()+"时");
        System.out.println(dateTime.getMinute()+"分");
        System.out.println(dateTime.getSecond()+"秒");
        System.out.println(dateTime.toString());
    }
}
```

DateTimeFormatter类

DateTimeFormatter类用于将字符串解析为日期

常用方法：

1. `static ofPattern(String pattern);`
作用：按pattern字符串指定的格式创建DateTimeFormatter对象
2. `LocalDateTime.parse(strDate, formatter);`
作用：按指定模板和字符串创建LocalDateTime对象

案例

将字符串“2014-04-01:13:24:01”格式化为一个LocalDateTime类的对象，并显示年、月、日、时、分和秒

```
public class Test_DateTimeFormatter{  
    public static void main(String[] args){  
        DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd:HH:mm:ss");  
        LocalDateTime date = LocalDateTime.parse("2015-04-06:00:00", dateTimeFormatter);  
    }  
}
```

ZonedDateTime 类

ZonedDateTime类处理日期和时间与相应的时区

常用方法：

1. ZonedDateTime.now()
获取系统当前日期和事件
2. String format(DateTimeFormatter formatter)
按指定模板将日期对象格式化为一个字符串。

案例

将当前日期格式化为字符串并显示年、月、日、时、分和秒

```
public class Test_ZonedDateTime{  
    public static void main(String[] args){  
        ZonedDateTime date = ZonedDateTime.now();  
        DateTimeFormatter formatter = DateTimeFormatter.OfPattern("M  
        String strDate = date.format(formatter);  
        System.out.println(strDate);  
    }  
}
```