# Jdk8重要的新功能

## Lambda表达式

语法：(Parameters) -> { Body } ->为jdk新加的操作符，用来将Lambda表达式拆分成两部分

左侧参数，右侧lambad体，是实现接口的功能，可以用lambda表达式实现函数式接口

Runnable task=()->{

System.out.println("进入任务");

};

作用域：可访问 static 修饰的成员变量可以改值。可访问表达式外层的局部变量不可改值。

## Stream

主要对集合操作的类。大部分情况下stream是容器调用Collection.stream()方法得到的，但stream和collections有以下不同：

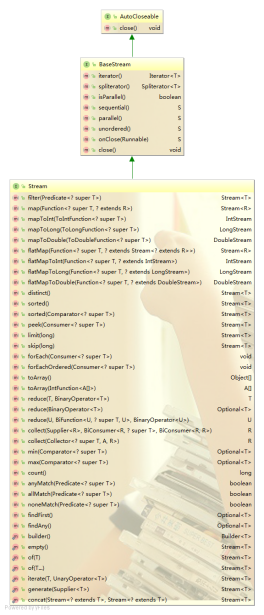
无存储：stream不是一种数据结构，它只是某种数据源的一个视图，数据源可以是一个数组，Java容器或I/O channel等。

为函数式编程而生：对stream的任何修改都不会修改背后的数据源，比如对stream执行过滤操作并不会删除被过滤的元素，而是会产生一个不包含被过滤元素的新stream。

惰式执行：stream上的操作并不会立即执行，只有等到用户真正需要结果的时候才会执行。

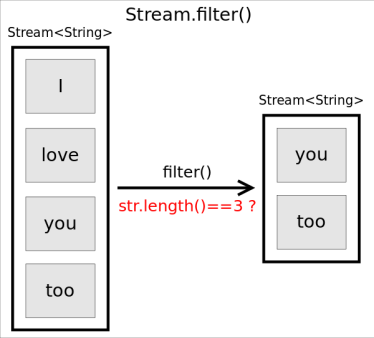
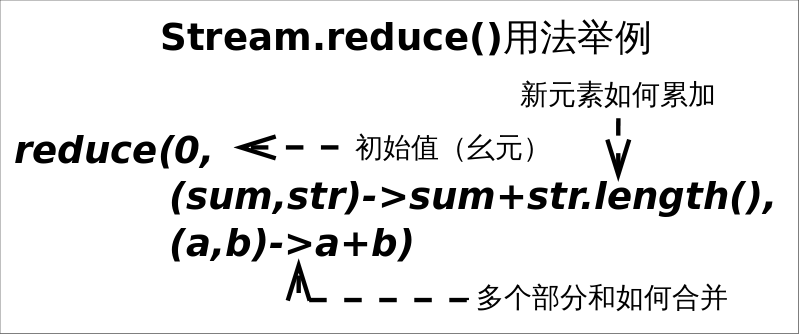
可消费性：stream只能被“消费”一次，一旦遍历过就会失效，就像容器的迭代器那样，想要再次遍历必须重新生成。即当执行Stream的void方法后失效

结构图

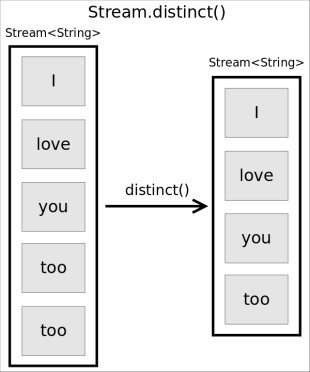
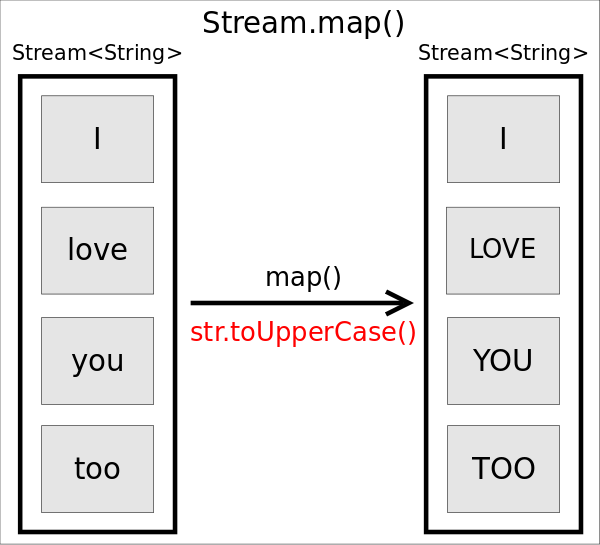
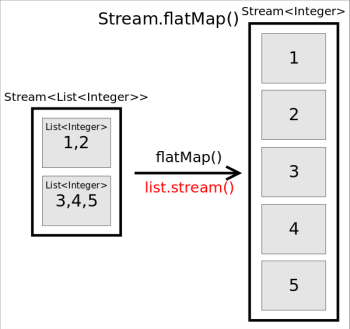


几个常用方法





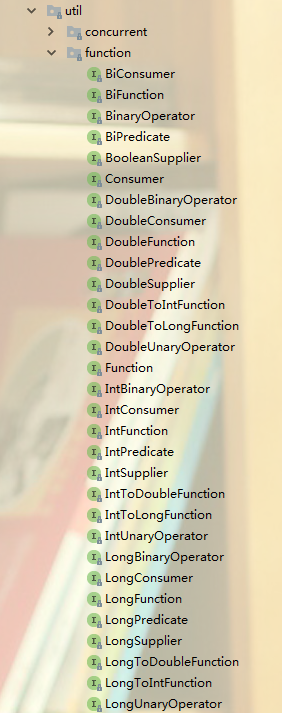
  



分组、最大值、最小值、拼接...很多功能。

## 函数式接口

只有一个非覆盖抽象方法，可用@FunctionalInterface限制（主要是配合lambda使用）



|  |
| --- |
| interface Function<T, R> {  R apply(T t);  default <V> Function<V, R> compose(Function<? super V, ? extends T> before) {  Objects.requireNonNull(before);  return (V v) -> apply(before.apply(v));  }  default <V> Function<T, V> andThen(Function<? super R, ? extends V> after) {  Objects.requireNonNull(after);  return (T t) -> after.apply(apply(t));  }  static <T> Function<T, T> identity() {  return t -> t;  } } |

## **[接口的默认方法与静态方法](https://www.cnblogs.com/JimKing/p/9155096.html)**

JDK8起，允许我们在interface中使用static和default修饰方法（使用这两种修饰符中其一就不能使用abstract修饰符），从而方法具有方法体

如：

|  |
| --- |
| @FunctionalInterface public interface Comparator<T> {  int compare(T o1, T o2);  boolean equals(Object obj);  default java.util.Comparator<T> reversed() {  return Collections.reverseOrder(this);  }  public static <T extends Comparable<? super T>> java.util.Comparator<T> reverseOrder() {  return Collections.reverseOrder();  } } |

## 方法引用

与Lambda表达式联合使用方法引用是用来直接访问类或者实例的已经存在的方法或者构造方法。方法引用提供了一种引用而不执行方法的方式，它需要由兼容的函数式接口构成的目标类型上下文。计算时，方法引用会创建函数式接口的一个实例。

当Lambda表达式中只是执行一个方法调用时，不用Lambda表达式，直接通过方法引用的形式可读性更高一些。方法引用是一种更简洁易懂的Lambda表达式。

注意方法引用是一个Lambda表达式，其中方法引用的操作符是双冒号"::"。

Runnable runnable= System.out::println;

## 注解扩展

类型注解：在java 8之前，注解只能是在声明的地方所使用，java8开始，注解可以应用在任何地方

创建类实例

new@Interned MyObject();

类型映射

myString = (@NonNull String) str;

implements 语句中

class UnmodifiableList<T> implements@Readonly List<@Readonly T> { ... }

throw exception声明

void monitorTemperature() throws@Critical TemperatureException { ... }

Note：

在Java 8里面，当类型转化甚至分配新对象的时候，都可以在声明变量或者参数的时候使用注解。

Java注解可以支持任意类型。

类型注解只是语法而不是语义，并不会影响java的编译时间，加载时间，以及运行时间，也就是说，编译成class文件的时候并不包含类型注解。

2）新增ElementType.TYPE\_USE 和ElementType.TYPE\_PARAMETER（在Target上）

新增的两个注释的程序元素类型 ElementType.TYPE\_USE 和 ElementType.TYPE\_PARAMETER用来描述注解的新场合。

ElementType.TYPE\_PARAMETER 表示该注解能写在类型变量的声明语句中。

ElementType.TYPE\_USE 表示该注解能写在使用类型的任何语句中（eg：声明语句、泛型和强制转换语句中的类型）。

eg：

@Target({ElementType.TYPE\_PARAMETER, ElementType.TYPE\_USE})

@interface MyAnnotation {}

3)类型注解的作用

类型注解被用来支持在Java的程序中做强类型检查。配合第三方插件工具Checker Framework（注：此插件so easy,这里不介绍了），可以在编译的时候检测出runtime error（eg：UnsupportedOperationException； NumberFormatException；NullPointerException异常等都是runtime error），以提高代码质量。这就是类型注解的作用。

重复注解

允许在同一声明类型（类，属性，或方法）上多次使用同一个注解。

Java8以前的版本使用注解有一个限制是相同的注解在同一位置只能使用一次，不能使用多次。

Java 8 引入了重复注解机制，这样相同的注解可以在同一地方使用多次。重复注解机制本身必须用 @Repeatable 注解。

实际上，重复注解不是一个语言上的改变，只是编译器层面的改动，技术层面仍然是一样的。

自定义一个包装类Hints注解用来放置一组具体的Hint注解

@interface MyHints {

Hint[] value();

}

@Repeatable(MyHints.class)

@interface Hint {

String value();}

## 最新的Date/Time API (JSR 310)

## 新增base64加解密API

|  |
| --- |
| BASE64严格的说，属于编码格式而非加密算法 作用是将人肉眼可以识别的信息，转换为不可以识别的数据，常用在http请求中。  1、加密（最基本的）  /\*\*  \* 将密码 加密  \* @param password 密码明文  \* @param charset 编码格式  \* @return 加密后的密码  \*/  public static String encryptByBase64\_jdk8(String password, String charset) {// UTF-8  String encryptPwd\_jdk8 = null;  try {  encryptPwd\_jdk8 = Base64.getEncoder().encodeToString(password.getBytes(charset));  } catch (UnsupportedEncodingException e) {  e.printStackTrace();  }  return encryptPwd\_jdk8;  }  2、解密  /\*\*  \* 将加密后的密码解密  \* @param encryptPwd 加密后的密码  \* @param charset 编码格式  \* @return 密码明文  \*/  public static String decryptByBase64\_jdk8(String encryptPwd, String charset) {// UTF-8  String password = null;  byte[] bytes = Base64.getDecoder().decode(encryptPwd);  try {  password = new String(bytes, charset);  } catch (UnsupportedEncodingException e) {  e.printStackTrace();  }  return password;  } |

## 数组并行（parallel）操作

数组排序，一般使用Arrays.sort()方法串行排序，Java8新增方法Arrays.parallelSort()并行排序。对数据量大的时候速度效率极大的提升

## JVM的PermGen空间被移除

取代它的是Metaspace（JEP 122）元空间