java8新特性

1. 接口定义增强

jdk1.8之前，接口的定义： 全局的静态常量 + 抽象方法

问题：某个接口有上千个实现，接口突然要新增一个方法，且所有子类的实现完全一样=====》

怎么办？ 每个子类都要实现一遍。

jdk1.8之后，接口的定义： 增加普通方法 default 和静态方法 static 的定义

2. 函数式接口

2.1 两种实现方式：

2.1.1 在接口中定义唯一一个抽象方法

2.1.2 接口上使用 @FunctionalInterface

两者并不冲突：

-- 注解只是强制约束，只能存在一个抽象方法

-- 唯一一个抽象方法：可以存在static方法和default普通方法

函数式接口中允许定义Object类的public方法；

例如：

```
@FunctionalInterface
public interface Comparator<T> {
}
```

```
@FunctionalInterface
public interface Runnable {
    public abstract void run();
}
```

抽象类和接口

NO.	比较点	抽象类	接口
1	定义	用abstract修饰的类。	用interface修饰的类
2	组成	抽象方法，普通方法，构造方法，成员变量，常量	抽象方法，静态常量
3	使用	子类继承(extends)	类实现(implements)
4	关系	抽象类可以实现接口	接口不能继承抽象类
5	对象	都是通过对象的多态性来实现的	
6	局限	不能多继承	可以多继承
7	选择	建议选择接口，避免单继承。	

```
Java类之间并不允许多继承，只可以单继承和实现多接口
一个类只能extends一个父类，但可以implements多个接口。
一个接口则可以同时extends多个接口，却不能implements任何接口。
===>>Java中的接口是支持多继承的。

public interface ApplicationContext extends EnvironmentCapable, ListableBeanFactory,
HierarchicalBeanFactory,
      MessageSource, ApplicationEventPublisher, ResourcePatternResolver {

      }
```

3. Lamda表达式

为什么引入lamda表达式?

java枚举类出现的目的---》解决多例设计模式的问题

3.1引入lamda表达式的目的 减少匿名内部类的使用

-----》解决匿名内部类代码丑陋，臃肿的问题

匿名内部类用法？？？补

3.2 lamda表达式核心语法:

3.2.1 (参数列表) —> 单行语句

```
FuncInterfaceDefination demo3 = (a) -> a;// 单行
```

3.2.2 (参数列表) —> { 多行语句}

```
FuncInterfaceDefination demo2 = (content) -> { // 多行
    return content;
};
```

3.2.3 (参数列表) —> 表达式

```
FuncInterfaceDefination demo3 = (a) -> a + a;// 表达式
```

3.3 方法引用

前提：引用函数式接口的方法

3.3.1 引用静态方法

类名称 :: static 方法名称

3.3.2 引用某个对象的普通方法

实例化对象 :: 普通方法

3.3.3 引用特定类型的方法

特定类 :: 方法（普通方法）

3.3.4 引用构造方法

类名称 :: new

4. 内建函数式接口

`jdk提供的 \Java\jdk1.8.0_171\jre\lib\rt.jar!\java\util\function`

4.1 完整功能型接口 (Function) 有参数有返回值

```
@FunctionalInterface
public interface Function<T, R> {
    R apply(T t);
}
```

4.2 消费型接口 (Consumer) 只接受参数无返回值

```
@FunctionalInterface
public interface Consumer<T> {
    void accept(T t);
}
```

4.3 供给型接口 (Supplier) 只返回结果无需参数

```
@FunctionalInterface
public interface Supplier<T> {
    T get();
}
```

4.4 断言型接口 (Predicate) 需参数返回true/false

```java
@FunctionalInterface
public interface Predicate<T> {
    boolean test(T t);
}
```

5. 集合的流式操作

6.Stream

`parallelStream`并行流

7.parallel 数组 并行数组

一般`Arrays.sort()`方法串行排序

Java8 新增方法`Arrays.parallelSort()`并行排序

8.Base64

9.LocalDateTime