Java 8

Новое в Java 8

Everything about Java 8

Book: What's New in Java 8

JAVA 8 FEATURES

Видео:

JDK8: Я, лямбда (Сергей Куксенко, jeeconf-2013)

Сергей Куксенко — Stream API, часть 1

Сергей Куксенко — Stream API, часть 2

Выбор в пользу использования интерфейсов вместо добавления в Java типов функций был преднамеренным.

Это устраняет необходимость во внесении значительных изменений в Java-библиотеки,

а также позволяет использовать лямбда-выражения с существующими библиотеками.

Оборотная сторона этого подхода состоит в том, что он ограничивает Java 8 так называемым "интерфейсным программированием" или функционально-подобным программированием — вместо истинного функционального программирования.

Default and static method in Interface

```
public interface Iterator<E>
 default void remove() { throw new UnsupportedOperationException("remove"); }
 default void forEachRemaining(Consumer<? super E> action) {
   Objects.requireNonNull(action);
   while (hasNext())
       action.accept(next());
 }
public interface Iterable<T> {
   default void forEach(Consumer<? super T> action) {
```

Differ from abstract class: used in mix-in and lambda

If ambiguous compiler will throw exception: need to provide implementation in class

Collections API additions

`Iterator default method forEachRemaining(Consumer action)`

`Iterable default method forEach(Consumer action)`

`Collection default method removeIf(Predicate filter)`

`Map replaceAll(), compute(), merge()`

Comparator

`reversed, thenComparing, naturalOrder, nullsFirst/nullsLast, comparing(Function<? super T, ? extends U> keyExtractor`

forEach() vs foreach loop

Lambda (o1, o2) -> o1.compareTo(o2)

Functional interface

```
final Predicate<String> p1 = e -> e.startsWith("a");
Runnable r = () -> System.out.println("Thread");
```

Лямбда-выражения

method reference: `object::instanceMethod, Class::staticMethod, Class::instanceMethod, EnclosingClass.this::method`

```
final Predicate<String> p2 = Objects::isNull;
final Predicate<String> p3 = String::isEmpty;
```

constructor reference: `Button::new`

```
Stream<Button> stream = list.stream().map(Button::new);
List<Button> buttons = stream.toArray(Button[]::new)
```

IDEA: RuntimeStatisticsPanel sample (Ctrl+Alt+V, Alt+Enter, Alt+Ctrl+N)

Implemented in JVM via invokedynamic

Limits: Can't use non-final variables (effectively final)

Can't handle checked exceptions

Limited flow-control (continue=break, no break, return)

```
Comparator<Integer> cmp = (a, b) -> {
   int x = a;
   int y = b;
   return Integer.compare(x, y);
};
Comparator<Integer> cmp = Integer::compare
```

this - вложенный класс

hiding переменных запрещен

Стандартные функциональные интерфейсы (java.util.function)

Function<T, R> - take a T as input, return an R as ouput

Predicate<T> - take a T as input, return a boolean as output

Consumer<T> - take a T as input, perform some action and don't return anything

Supplier<T> - with nothing as input, return a T

BinaryOperator<T> - take two T's as input, return one T as output, useful for "reduce" operations

Primitive specializations for most of these exist as well. They're provided in int, long, and double forms.

IntConsumer - take an int as input, perform some action and don't return anything

```
final List<String> list = new LinkedList<>(Arrays.asList("a","ab", "cb", "c"));
final Predicate<String> pred1 = e -> e.startsWith("a");
list.removeIf(pred1.and(e -> e.endsWith("b")));
list.removeIf(((Predicate<String>) e -> e.startsWith("a")).and(e ->
e.endsWith("b")));
```

Replace Guava

Replace for LoadingCache

Java 8 Stream Tutorial

Stream: like an iterator, can only be traversed once, may also be infinite)

Stream<T> of(T... values), Stream.Builder, collection.stream()

S parallel();

Stream<T> filter(Predicate<? **super** T> predicate);

<R> Stream<R> map(Function<? **super** T, ? **extends** R> mapper);

<R> Stream<R> flatMap(Function<? **super** T, ? **extends** Stream<? **extends** R>> mapper);

Optional<T> reduce(BinaryOperator<T> accumulator) / min, max, sum(IntStream), count

void forEach(Consumer<? **super** T> action) / forEachOrdered

R collect(Collectors.toList()), A[] toArray()

There are primitive-specialized versions of Stream for ints, longs, and doubles:

- IntStream
- LongStream
- DoubleStream

...Stream mapTo...(To...Function<? **super** T> mapper);

Intermediate (lazy) operations:

- **filter** 1 - Exclude all elements that don't match a Predicate.
- **map** 1 2 3 4 - Perform a one-to-one transformation of elements using a Function.
- **flatMap** 1 2 3 4 - Transform each element into zero or more elements by way of another Stream.
- **peek** 1 - Perform some action on each element as it is encountered. Primarily useful for debugging.
- **distinct** 1 - Exclude all duplicate elements according to their .equals behavior. This is a stateful operation.
- **sorted** 1 2 - Ensure that stream elements in subsequent operations are encountered according to the order imposed by a Comparator. This is a stateful operation.
- **limit** 1 - Ensure that subsequent operations only see up to a maximum number of elements. This is a stateful, short-circuiting operation.
- **skip** 1 - Ensure that subsequent operations do not see the first n elements. This is a stateful operation.

Terminal operations:

- **forEach** 1 - Perform some action for each element in the stream.
- **toArray** 1 2 - Dump the elements in the stream to an array.
- **reduce** 1 2 3 - Combine the stream elements into one using a BinaryOperator.
- **collect** 1 2 - Dump the elements in the stream into some container, such as a Collection or Map.
- **min** 1 - Find the minimum element of the stream according to a Comparator.
- **max** 1 - Find the maximum element of the stream according to a Comparator.
- **count** 1 - Find the number of elements in the stream.
- **anyMatch** 1 - Find out whether at least one of the elements in the stream matches a Predicate. This is a short-circuiting operation.
- **allMatch** 1 - Find out whether every element in the stream matches a Predicate. This is a short-circuiting operation.
- **noneMatch** 1 - Find out whether zero elements in the stream match a Predicate. This is a short-circuiting operation.
- **findFirst** 1 - Find the first element in the stream. This is a short-circuiting operation.
- **findAny** 1 - Find any element in the stream, which may be cheaper than findFirst for some streams. This is a short-circuiting operation.

Collection->Stream: Stream<E> Collection.stream()

```
default Stream<E> stream() {
   return StreamSupport.stream(spliterator(), false);
}

default Stream<E> parallelStream() {
   return StreamSupport.stream(spliterator(), true);
}
```

Spliterator: an wpapper for traversing and partitioning elements of a source

```
default Spliterator<E> spliterator() {
   return Spliterators.spliterator(this, 0);
}

Spliterators.IteratorSpliterator(Collection<? extends T> collection, int characteristics) {
   this.collection = collection;
   this.it = null;
   this.characteristics = ...
```

Stream->Iterator

```
Iterable<Integer> iterator = IntStream.range(0, 10)::iterator;
for (final int i : iterator) // 2d time: IllegalStateException: stream has already been operated upon or closed
IntStream.range(0, 10).forEach(System.out::println);
```

CompletableFuture: collects all the features of ListenableFuture in Guava with SettableFuture.

Creating and obtaining CompletableFuture

```
static <U> CompletableFuture<U> supplyAsync(Supplier<U> supplier);
static CompletableFuture<Void> runAsync(Runnable runnable);
```

Transforming and acting on one CompletableFuture

```
<U> CompletableFuture<U> thenApply(Function<? super T,? extends U> fn);
<U> CompletableFuture<U> thenApplyAsync(Function<? super T,? extends U> fn);
```

(Async apply it asynchronously in different thread pool)

```
f1.thenApply(Integer::parseInt).thenApply(r -> r * r * Math.PI)
```

Explicitly completed (setting its value and status) or running code on completion

Error handling of single CompletableFuture

Combining CompletableFuture (two or array) together

Concurrency API additions

ForkJoinPool.commonPool()
ConcurrentHashMap

- ConcurrentHashMap.reduce...
- ConcurrentHashMap.search...
- ConcurrentHashMap.forEach.

Generic type inference improvements

Utility.<Type>foo().bar();

IO/NIO API additions

Files: UTF-8 default and Stream: Stream<String> lines = Files.lines(Paths.get(“read.me));

BufferedReader.lines()

Math ..Exact: throw new ArithmeticException("integer overflow");

String.join, StringJoiner

Base64

@Repeatable Annotatios.

Time API (java.time)

Date and Time API changes in Java 8

Java 8 Date Time API (java.time) vs Joda-Time

Java 8 classes are built around the human time/ Joda-Time is using machine time inside
No MutableDateTime
No Null
Enum DayOfWeek and Month
More timezone features
No Interval support