HowToDoInJava

Get all Dates between Two Dates as Stream

Last Updated: March 3, 2022 By: Lokesh Gupta Java 9 Java Date Time, Java Stream Basics

Date and time handling has always been a pain area for Java developers. The new Date-Time API added in Java 8 changed the way, we interact with date and time in Java.

New Date API is a very powerful and much-needed improvement. The only thing missing was, **getting a stream of dates** having some common difference between two subsequent dates (though it was possible there was no easy way).

Java 9 has introduced a new method `LocalDate.datesUntil()` that can give a stream on dates. Using `datesUntil()` makes it easy to **create dates streams with a fixed offset**.

Table Of Contents

1. LocalDate.datesUntil() Method (Java 9)

1.1. Syntax

This method has two overloaded forms:

- `startDate.datesUntil(endDate)` : returns a sequential ordered stream of dates that starts from `startDate` (*inclusive*) and goes to `endDate` (*exclusive*) by an incremental step of 1 day.
- `startDate.datesUntil(endDate, period)` : same as above with a configured incremental step `period`.

```
Stream<LocalDate> datesUntil(LocalDate end)
Stream<LocalDate> datesUntil(LocalDate end, Period step)
```

1.2. Example of Stream of Dates

Creating a stream of dates is very simple and straightforward as demonstrated in the given examples.

In this example, we are getting the dates for the next 3 consecutive days.

```
LocalDate today = LocalDate.now();

Stream<LocalDate> next3Days = today.datesUntil(today.plusDays(3));

next3Days.forEach(System.out::println);
```

In the next example, we are getting the same day for the next 3 weeks.

```
Stream<LocalDate> sameDayNext3Weeks = today
    .datesUntil(today.plusDays(21), Period.ofWeeks(1));

sameDayNext3Weeks.forEach(System.out::println);
```

2. Get Stream of Dates using Iteration (Java 8)

If you have still not adapted Java 9, then you can use the given below method to generate date streams.

```
Stream<LocalDate> nextThreeDays = Stream.iterate(today, d -> d.plusDays(1));
```

Once we have the stream, we can use the stream operations on the items.

```
Stream<LocalDate> nextThreeDays = Stream.iterate(today, d -> d.plusDays(1));

List<LocalDate> list = nextThreeDays
    .limit(3)
    .collect(Collectors.toList());
```

Happy Learning !!

Sourcecode on Github

Was this post helpful?

Let us know if you liked the post. That's the only way we can improve.

Yes

No

Recommended Reading:

1. Getting All Dates Between Two Dates in Java
2. Java – Difference Between Two Dates
3. Find all Business Days between Two Dates
4. Get Number of Days between Two Dates
5. Comparing Two Dates in Java
6. Java regex to check invalid dates
7. Java Stream reuse – traverse stream multiple times?
8. Convert between Stream and Array
9. Java 9 Stream API Improvements
0. Convert between LocalDateTime and ZonedDateTime

Join 7000+ Awesome Developers

Get the latest updates from industry, awesome resources, blog updates and much more.

Email Address

Subscribe

** We do not spam !!*

2 thoughts on “Get all Dates between Two Dates as Stream”

Amol

November 9, 2019 at 2:42 pm

Hi Lokesh,

I want to create a sequential date time generator which is required for performance testing within a boundary of start date and end date.
For e.g. Start Date: 2018-11-01 00:00:00 End Date: 2018-11-10 23:59:59

1st Value= 2018-11-01 00:00:00
2nd Value = 2018-11-01 00:00:01
3rd Value = 2018-11-01 00:00:02
–
–
–
61st Value = 2018-11-01 00:01:00
62nd Value = 2018-11-01 00:01:01

How do I create this using Java 8?
Can you help me with this?

Thanks,
Amol

Reply

Lokesh Gupta

November 9, 2019 at 10:30 pm

Try editing this program as per your need.

```
import java.time.LocalDateTime;
import java.time.temporal.ChronoUnit;
import java.util.Iterator;
import java.util.stream.Stream;

public class Main
{
  public static void main(String[] args)
  {
    DateTimeRange range = new DateTimeRange(LocalDateTime.now(),
        LocalDateTime.now().plusDays(1));

    range.stream().forEach(System.out::println);
  }
}

class DateTimeRange
    implements Iterable&lt;LocalDateTime&gt;
{
  private final LocalDateTime startDateTime;
  private final LocalDateTime endDateTime;

  public DateTimeRange(LocalDateTime sdt,
        LocalDateTime edt) {
    this.startDateTime = sdt;
    this.endDateTime = edt;
  }

  @Override
  public Iterator&lt;LocalDateTime&gt; iterator() {
    return stream().iterator();
  }

  public Stream&lt;LocalDateTime&gt; stream()
  {
    return Stream.iterate(startDateTime, d -&gt; d.plusSeconds(1))
      .limit(ChronoUnit.SECONDS.between(startDateTime, endDateTime) + 1);
  }
}
```

Reply

Leave a Comment

Name *

Email *

Website

Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search …

HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- About Me
- Contact Us
- Privacy policy
- Advertise
- Guest Posts

Blogs

REST API Tutorial