



[Home](#) » [Java](#) » [Core Java](#) » [Java Best Practices – DateFormat in a Multithreading Environment](#)

ABOUT BYRON KIOURTZOGLU



Byron is a master software engineer working in the IT and Telecom domains. He is an applications developer in a wide variety of applications/services. He is currently acting as the team leader and technical architect for a proprietary service creation and integration platform for both the IT and Telecom industries in addition to a in-house big data real-time analytics solution. He is always fascinated by SOA, middleware services and mobile development. Byron is co-founder and Executive Editor at Java Code Geeks.



Java Best Practices – DateFormat in a Multithreading Environment

Posted by: [Byron Kiourtzoglou](#) in [Core Java](#) July 11th, 2010 10 Comments 2609 Views

This is the first of a series of articles concerning proposed practices while working with the Java programming language.

All discussed topics are based on use cases derived from the development of mission critical, ultra high performance production systems for the telecommunication industry.

Prior reading each section of this article it is highly recommended that you consult the relevant Java API documentation for detailed information and code samples.

All tests are performed against a Sony Vaio with the following characteristics :

- System : openSUSE 11.1 (x86_64)
- Processor (CPU) : Intel(R) Core(TM)2 Duo CPU T6670 @ 2.20GHz
- Processor Speed : 1,200.00 MHz
- Total memory (RAM) : 2.8 GB
- Java : OpenJDK 1.6.0_0 64-Bit

The following test configuration is applied :

- Concurrent worker Threads : 200
- Test repeats per worker Thread : 1000
- Overall test runs : 100

Using DateFormat in a multithreading environment

Working with DateFormat in a multithreading environment can be tricky. The Java API documentation clearly states :

"Date formats are not synchronized. It is recommended to create separate format instances for each thread. If multiple threads access a format concurrently, it must be synchronized externally."

A typical case scenario is to convert a Date to its String representation or vice versa, using a predefined format. Creating new DateFormat instances for every conversion is very inefficient. You should keep in mind that the static factory methods "getDateInstance(..)" also create new DateFormat instances when used. What most developers do is that they construct a DateFormat instance, using a DateFormat implementation class (e.g. SimpleDateFormat), and assign its value to a class variable. The class scoped variable is used for all their Date parsing and formatting needs. The aforementioned approach, although very efficient, can cause problems when multiple threads access the same instance of the class variable, due to lack of synchronization on the DateFormat class. Typical exceptions thrown when parsing to create a Date object are :

- java.lang.NumberFormatException
- java.lang.ArrayIndexOutOfBoundsException

You should also experience malformed Date to String representation when formatting is performed.

NEWSLETTER

Insiders are already enjoying weekly up-to-date complimentary whitepapers!

Join them now to gain **exclusive access** to the latest news in the Java world as well as insights about Android, Scala, and other related technologies.

☐ I agree to the Terms and Privacy Policy

JOIN US



With **1,240,600** unique visitors and **500** authors placed among related sites and constantly being looked out for par excellence you So If you have unique and interesting content then you check out our **JCG** partners program. You can be a **guest writer** for Java Code Geek and showcase your writing skills!

To properly handle the aforementioned issues, it is vital to clarify the architecture of your multithreading environment. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently. Typically, in a multithreading environment (either a container inside the JVM or the JVM itself), Thread pooling should be performed. Worker threads should be constructed and initialized upon startup, utilized to execute your programs. For example a Web container constructs a pool of worker threads to serve all incoming traffic. Thread pooling is the most efficient way to manipulate system resources mainly due to the fact that Thread creation and initialization is a high resource consuming task for the Java Virtual Machine. Nevertheless application parallelism can be achieved by simply creating a new Thread of execution for every piece of code you want to be executed concurrently.

Concerning class scoped DateFormat instances :

- If you have clarified that **NO** Thread pools are used in your environment then only new Thread instances concurrently access your DateFormat instance. In this case it is recommended to synchronize that DateFormat instance externally
- In case Thread pools are used, there is a limited number of Thread instances that can access your DateFormat instance concurrently. Thus it is recommended to create separate DateFormat instances for each thread using the ThreadLocal approach

Below are examples of "getDateInstance(..)", "synchronization" and ThreadLocal approaches :

```
01 package com.javacodegeeks.test;
02
03 import java.text.DateFormat;
04 import java.text.ParseException;
05 import java.text.SimpleDateFormat;
06 import java.util.Date;
07
08 public class ConcurrentDateFormatAccess {
09
10     public Date convertStringToDate(String dateString) throws ParseException {
11         return SimpleDateFormat.getDateInstance(DateFormat.MEDIUM).parse(dateString);
12     }
13
14 }
```

```
01 package com.javacodegeeks.test;
02
03 import java.text.DateFormat;
04 import java.text.ParseException;
05 import java.text.SimpleDateFormat;
06 import java.util.Date;
07
08 public class ConcurrentDateFormatAccess {
09
10     private DateFormat df = new SimpleDateFormat("yyyy MM dd");
11
12     public Date convertStringToDate(String dateString) throws ParseException {
13         Date result;
14         synchronized(df) {
15             result = df.parse(dateString);
16         }
17         return result;
18     }
19
20 }
```

Things to notice here :

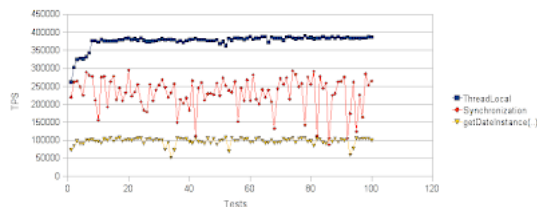
- Every individual Thread executing the "convertStringToDate" operation, is trying to acquire the monitor lock on the DateFormat Object prior acquiring a reference to the DateFormat class variable instance . If another Thread is holding the lock then the current Thread waits until the lock is released. That way only one Thread is accessing the DateFormat instance at a time

```
01 package com.javacodegeeks.test;
02
03 import java.text.DateFormat;
04 import java.text.ParseException;
05 import java.text.SimpleDateFormat;
06 import java.util.Date;
07
08 public class ConcurrentDateFormatAccess {
09
10     private ThreadLocal<DateFormat> df = new ThreadLocal<DateFormat> () {
11
12         @Override
13         public DateFormat get() {
14             return super.get();
15         }
16
17         @Override
18         protected DateFormat initialValue() {
19             return new SimpleDateFormat("yyyy MM dd");
20         }
21
22         @Override
23         public void remove() {
24             super.remove();
25         }
26
27         @Override
28         public void set(DateFormat value) {
29             super.set(value);
30         }
31     };
32
33     public Date convertStringToDate(String dateString) throws ParseException {
34         return df.get().parse(dateString);
35     }
36
37 }
```

Things to notice here :

- Every individual Thread executing the "convertStringToDate" operation, invokes the "df.get()" operation in order to initialize or retrieve an already initialized reference of its local scoped DateFormat instance

Below we present a performance comparison chart between the three aforementioned approaches (notice that we have tested the parsing functionality of the DateFormat utility class. We convert a String representation of a date to its Date Object equivalent, according to a specific date format).



The horizontal axis represents the number of test runs and the vertical axis the average transactions per second (TPS) for each test run. Thus higher values are better. As you can see by using Thread pools and the ThreadLocal approach you can achieve superior performance compared to the "synchronization" and the "getDateInstance(...)" approaches.

Lastly, let me pinpoint that using the ThreadLocal approach without Thread pools, is equivalent to using the "getDateInstance(...)" approach due to the fact that every new Thread has to initialize its local DateFormat instance prior using it, thus a new DateFormat instance will be created with every single execution.

Happy Coding!

Justin

Related Articles :

- Java Best Practices – High performance Serialization
- Java Best Practices – Vector vs ArrayList vs HashSet
- Java Best Practices – String performance and Exact String Matching
- Java Best Practices – Queue battle and the Linked ConcurrentHashMap
- Java Best Practices – Char to Byte and Byte to Char conversions

Tagged with: [DATE](#) [DATEFORMAT](#) [JAVA BEST PRACTICES](#) [MULTITHREADING](#)

(0 rating, 0 votes)

You need to be a registered member to rate this. 10 Comments 2609 Views Tweet it!

Do you want to know how to develop your skillset to become a **Java Rockstar**?



Subscribe to our newsletter to start Rocking right now!
To get you started we give you our best selling eBooks for **FREE!**

1. JPA Mini Book
2. JVM Troubleshooting Guide
3. JUnit Tutorial for Unit Testing
4. Java Annotations Tutorial
5. Java Interview Questions
6. Spring Interview Questions
7. Android UI Design

and many more

☐ I agree to the Terms and Privacy Policy

LIKE THIS ARTICLE? READ MORE FROM JAVA CODE GEEKS



Join the discussion...



9 1 0 ⚡ 🔥

10

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)

✉ Subscribe ▾

▲ newest ▲ oldest ▲ most voted



yc y



Could u please provide the source code and the testing code?

Guest

+ 0 - [Reply](#)

🕒 6 years ago



Jangkoon, Hwang



What's better of the two?

- 1) ThreaLocal approach with SimpleDateFormat
- 2) FastDateFormat

Guest

+ 0 - [Reply](#)

🕒 5 years ago



Chas Honton



Commons Lang 3.x now has FastDateParser and FastDateFormat. These classes are thread safe and faster than SimpleDateFormat. They also the same format/parse pattern specifications as SimpleDateFormat.

Guest

+ 0 - [Reply](#)

🕒 4 years ago ▲



fito



Thanks Chas!

That's the solution, simplest and reliable.

Guest

+ 0 - [Reply](#)

🕒 4 years ago



radha



How does this compare with <http://commons.apache.org/proper/commons-lang/apidocs/org/apache/commons/lang3/time/FastDateFormat.html>, do you have a benchmark script to include commons API also?

Guest

+ 0 - [Reply](#)

🕒 4 years ago



Jo Desmet



The problem is not 'lack of synchronization'. Most Developers understand that for most classes that are not thread safe, that this is due to concurrently changing state. Once a Format is established, formatting a Date should not change state. Simply documenting this in official documentation as not thread-safe is not enough. It should be explicitly documented that even the format method is not thread-safe if it maintains temporary state in instance variables. Declaring it as static is not just a rookie mistake. Analogy can be made between modifying a collection (put) vs accessing a collection (get).

Guest

+ 0 - [Reply](#)

🕒 3 years ago



ali



thanks for sharing this, really i find some difficulties using date in java , this is why i prefer using joda time library ,

Guest



+ 0 - [Reply](#)

🕒 3 years ago





Guest

Octavio Bokel  

A code for creating an DateFormat using thread local. Each thread has only a DateFormat instance for a given pattern. If the pattern is not found, it uses SimpleDateFormat to create a new. Should be a more generic version of the code above. public class ThreadLocalDateFormatMap { protected static ThreadLocalDateFormatMap INSTANCE = new ThreadLocalDateFormatMap(); public static ThreadLocalDateFormatMap getInstance() { return INSTANCE; } protected ThreadLocal<Map> localDateFormatMap = new ThreadLocal<Map>() { protected Map initialValue() { return new HashMap(); } }; protected DateFormat createSimpleDateFormat(String pattern) { DateFormat result = new SimpleDateFormat(pattern); putDateFormat(pattern, result); return result; } public DateFormat putDateFormat(String pattern, DateFormat format) {... [Read more »](#)

+ 0 -

 Reply

🕒 3 years ago



Guest

Kri 

Can anyone please explain what does the following mean with respect to this article ? Will there be any problem if we use ThreadLocal approach.? "If one had used one of the application classes instead of the JDK bundled DateFormat classes loaded by the bootstrap classloader, we are already in the danger zone. Just forgetting to remove it after the task at hand is completed, a copy of that Object will remain with the Thread, which tends to belong to a thread pool. Since lifespan of the pooled Thread surpasses that of the application, it will prevent the object and... [Read more »](#)

+ 0 -

 Reply

🕒 2 years ago



Guest

Reiner Saddey 

It would be interesting to profile another approach: create static singleton, then use method scoped clone to do the parse / format.

+ 0 -

 Reply

🕒 5 months ago

KNOWLEDGE BASE

[Courses](#)

[Examples](#)

[Minibooks](#)

[Resources](#)

[Tutorials](#)

PARTNERS

[Mkyong](#)

THE CODE GEEKS NETWORK

[.NET Code Geeks](#)

[Java Code Geeks](#)

[System Code Geeks](#)

[Web Code Geeks](#)

HALL OF FAME

["Android Full Application Tutorial" series](#)

[11 Online Learning websites that you should check out](#)

[Advantages and Disadvantages of Cloud Computing – Cloud computing pros and cons](#)

[Android Google Maps Tutorial](#)

[Android JSON Parsing with Gson Tutorial](#)

[Android Location Based Services Application – GPS location](#)

[Android Quick Preferences Tutorial](#)

[Difference between Comparator and Comparable in Java](#)

[GWT 2 Spring 3 JPA 2 Hibernate 3.5 Tutorial](#)

[Java Best Practices – Vector vs ArrayList vs HashSet](#)

ABOUT JAVA CODE GEEKS

JCGs (Java Code Geeks) is an independent online community focused on creating the ultimate Java to Java developers resource center; targeted at the technical and non-technical team lead (senior developer), project manager and junior developer. JCGs serve the Java, SOA, Agile and Telecom communities with daily news, white domain experts, articles, tutorials, reviews, announcements, code snippets and source projects.

DISCLAIMER

All trademarks and registered trademarks appearing on Java Code Geeks are the property of their respective owners. Java is a trademark or registered trademark of Oracle Corporation in the United States and other countries. Examples Java Code Geeks is not connected to Oracle Corporation and is not sponsored by Oracle Corporation.

