# **[深入理解Java：SimpleDateFormat安全的时间格式化](http://www.cnblogs.com/peida/archive/2013/05/31/3070790.html)**

　　想必大家对SimpleDateFormat并不陌生。SimpleDateFormat 是 Java 中一个非常常用的类，该类用来对日期字符串进行解析和格式化输出，但如果使用不小心会导致非常微妙和难以调试的问题，因为 DateFormat 和 SimpleDateFormat 类不都是线程安全的，在多线程环境下调用 format() 和 parse() 方法应该使用同步代码来避免问题。下面我们通过一个具体的场景来一步步的深入学习和理解SimpleDateFormat类。

**一.引子**  
　　我们都是优秀的程序员，我们都知道在程序中我们应当尽量少的创建SimpleDateFormat 实例，因为创建这么一个实例需要耗费很大的代价。在一个读取数据库数据导出到excel文件的例子当中，每次处理一个时间信息的时候，就需要创建一个SimpleDateFormat实例对象，然后再丢弃这个对象。大量的对象就这样被创建出来，占用大量的内存和 jvm空间。代码如下：

[IMG_256](http://www.cnblogs.com/peida/archive/2013/05/31/javascript:void(0);)

package com.peidasoft.dateformat;

import java.text.ParseException;import java.text.SimpleDateFormat;import java.util.Date;

public class DateUtil {

public static String formatDate(Date date)throws ParseException{

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

return sdf.format(date);

}

public static Date parse(String strDate) throws ParseException{

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

return sdf.parse(strDate);

}

}

[IMG_257](http://www.cnblogs.com/peida/archive/2013/05/31/javascript:void(0);)

　　你也许会说，OK，那我就创建一个静态的simpleDateFormat实例，然后放到一个DateUtil类（如下）中，在使用时直接使用这个实例进行操作，这样问题就解决了。改进后的代码如下：

[IMG_258](http://www.cnblogs.com/peida/archive/2013/05/31/javascript:void(0);)

package com.peidasoft.dateformat;

import java.text.ParseException;import java.text.SimpleDateFormat;import java.util.Date;

public class DateUtil {

private static final SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

public static String formatDate(Date date)throws ParseException{

return sdf.format(date);

}

public static Date parse(String strDate) throws ParseException{

return sdf.parse(strDate);

}

}

[IMG_259](http://www.cnblogs.com/peida/archive/2013/05/31/javascript:void(0);)

　　当然，这个方法的确很不错，在大部分的时间里面都会工作得很好。但当你在生产环境中使用一段时间之后，你就会发现这么一个事实：它不是线程安全的。在正常的测试情况之下，都没有问题，但一旦在生产环境中一定负载情况下时，这个问题就出来了。他会出现各种不同的情况，比如转化的时间不正确，比如报错，比如线程被挂死等等。我们看下面的测试用例，那事实说话：

[IMG_260](http://www.cnblogs.com/peida/archive/2013/05/31/javascript:void(0);)

package com.peidasoft.dateformat;

import java.text.ParseException;import java.text.SimpleDateFormat;import java.util.Date;

public class DateUtil {

private static final SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

public static String formatDate(Date date)throws ParseException{

return sdf.format(date);

}

public static Date parse(String strDate) throws ParseException{

return sdf.parse(strDate);

}

}

[IMG_261](http://www.cnblogs.com/peida/archive/2013/05/31/javascript:void(0);)

[IMG_262](http://www.cnblogs.com/peida/archive/2013/05/31/javascript:void(0);)

package com.peidasoft.dateformat;

import java.text.ParseException;import java.util.Date;

public class DateUtilTest {

public static class TestSimpleDateFormatThreadSafe extends Thread {

@Override

public void run() {

while(true) {

try {

this.join(2000);

} catch (InterruptedException e1) {

e1.printStackTrace();

}

try {

System.out.println(this.getName()+":"+DateUtil.parse("2013-05-24 06:02:20"));

} catch (ParseException e) {

e.printStackTrace();

}

}

}

}

public static void main(String[] args) {

for(int i = 0; i < 3; i++){

new TestSimpleDateFormatThreadSafe().start();

}

}

}

[IMG_263](http://www.cnblogs.com/peida/archive/2013/05/31/javascript:void(0);)

　　执行输出如下：

[IMG_264](http://www.cnblogs.com/peida/archive/2013/05/31/javascript:void(0);)

Exception in thread "Thread-1" java.lang.NumberFormatException: multiple points

at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:1082)

at java.lang.Double.parseDouble(Double.java:510)

at java.text.DigitList.getDouble(DigitList.java:151)

at java.text.DecimalFormat.parse(DecimalFormat.java:1302)

at java.text.SimpleDateFormat.subParse(SimpleDateFormat.java:1589)

at java.text.SimpleDateFormat.parse(SimpleDateFormat.java:1311)

at java.text.DateFormat.parse(DateFormat.java:335)

at com.peidasoft.orm.dateformat.DateNoStaticUtil.parse(DateNoStaticUtil.java:17)

at com.peidasoft.orm.dateformat.DateUtilTest$TestSimpleDateFormatThreadSafe.run(DateUtilTest.java:20)

Exception in thread "Thread-0" java.lang.NumberFormatException: multiple points

at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:1082)

at java.lang.Double.parseDouble(Double.java:510)

at java.text.DigitList.getDouble(DigitList.java:151)

at java.text.DecimalFormat.parse(DecimalFormat.java:1302)

at java.text.SimpleDateFormat.subParse(SimpleDateFormat.java:1589)

at java.text.SimpleDateFormat.parse(SimpleDateFormat.java:1311)

at java.text.DateFormat.parse(DateFormat.java:335)

at com.peidasoft.orm.dateformat.DateNoStaticUtil.parse(DateNoStaticUtil.java:17)

at com.peidasoft.orm.dateformat.DateUtilTest$TestSimpleDateFormatThreadSafe.run(DateUtilTest.java:20)

Thread-2:Mon May 24 06:02:20 CST 2021

Thread-2:Fri May 24 06:02:20 CST 2013

Thread-2:Fri May 24 06:02:20 CST 2013

Thread-2:Fri May 24 06:02:20 CST 2013

[IMG_265](http://www.cnblogs.com/peida/archive/2013/05/31/javascript:void(0);)

　　说明：Thread-1和Thread-0报java.lang.NumberFormatException: multiple points错误，直接挂死，没起来；Thread-2 虽然没有挂死，但输出的时间是有错误的，比如我们输入的时间是：2013-05-24 06:02:20 ，当会输出：Mon May 24 06:02:20 CST 2021 这样的灵异事件。

**二.原因**

　　作为一个专业程序员，我们当然都知道，相比于共享一个变量的开销要比每次创建一个新变量要小很多。上面的优化过的静态的SimpleDateFormat版，之所在并发情况下回出现各种灵异错误，是因为SimpleDateFormat和DateFormat类不是线程安全的。我们之所以忽视线程安全的问题，是因为从SimpleDateFormat和DateFormat类提供给我们的接口上来看，实在让人看不出它与线程安全有何相干。只是在JDK文档的最下面有如下说明：

SimpleDateFormat中的日期格式不是同步的。推荐（建议）为每个线程创建独立的格式实例。如果多个线程同时访问一个格式，则它必须保持外部同步。

　　JDK原始文档如下：  
　　Synchronization：  
　　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.

　　下面我们通过看JDK源码来看看为什么SimpleDateFormat和DateFormat类不是线程安全的真正原因：

　　SimpleDateFormat继承了DateFormat,在DateFormat中定义了一个protected属性的 Calendar类的对象：calendar。只是因为Calendar累的概念复杂，牵扯到时区与本地化等等，Jdk的实现中使用了成员变量来传递参数，这就造成在多线程的时候会出现错误。

　　在format方法里，有这样一段代码：

[IMG_266](http://www.cnblogs.com/peida/archive/2013/05/31/javascript:void(0);)

private StringBuffer format(Date date, StringBuffer toAppendTo,

FieldDelegate delegate) {

// Convert input date to time field list calendar.setTime(date);

boolean useDateFormatSymbols = useDateFormatSymbols();

for (int i = 0; i < compiledPattern.length; ) {

int tag = compiledPattern[i] >>> 8;

int count = compiledPattern[i++] & 0xff;

if (count == 255) {

count = compiledPattern[i++] << 16;

count |= compiledPattern[i++];

}

switch (tag) {

case TAG\_QUOTE\_ASCII\_CHAR:

toAppendTo.append((char)count);

break;

case TAG\_QUOTE\_CHARS:

toAppendTo.append(compiledPattern, i, count);

i += count;

break;

default:

subFormat(tag, count, delegate, toAppendTo, useDateFormatSymbols);

break;

}

}

return toAppendTo;

}

[IMG_267](http://www.cnblogs.com/peida/archive/2013/05/31/javascript:void(0);)

　　calendar.setTime(date)这条语句改变了calendar，稍后，calendar还会用到（在subFormat方法里），而这就是引发问题的根源。想象一下，在一个多线程环境下，有两个线程持有了同一个SimpleDateFormat的实例，分别调用format方法：  
　　线程1调用format方法，改变了calendar这个字段。  
　　中断来了。  
　　线程2开始执行，它也改变了calendar。  
　　又中断了。  
　　线程1回来了，此时，calendar已然不是它所设的值，而是走上了线程2设计的道路。如果多个线程同时争抢calendar对象，则会出现各种问题，时间不对，线程挂死等等。  
　　分析一下format的实现，我们不难发现，用到成员变量calendar，唯一的好处，就是在调用subFormat时，少了一个参数，却带来了这许多的问题。其实，只要在这里用一个局部变量，一路传递下去，所有问题都将迎刃而解。  
　　这个问题背后隐藏着一个更为重要的问题--无状态：无状态方法的好处之一，就是它在各种环境下，都可以安全的调用。衡量一个方法是否是有状态的，就看它是否改动了其它的东西，比如全局变量，比如实例的字段。format方法在运行过程中改动了SimpleDateFormat的calendar字段，所以，它是有状态的。

　　这也同时提醒我们在开发和设计系统的时候注意下一下三点:

**1.自己写公用类的时候，要对多线程调用情况下的后果在注释里进行明确说明**

**2.对线程环境下，对每一个共享的可变变量都要注意其线程安全性**

**3.我们的类和方法在做设计的时候，要尽量设计成无状态的**

**三.解决办法**

**1.需要的时候创建新实例：**

[IMG_268](http://www.cnblogs.com/peida/archive/2013/05/31/javascript:void(0);)

package com.peidasoft.dateformat;

import java.text.ParseException;import java.text.SimpleDateFormat;import java.util.Date;

public class DateUtil {

public static String formatDate(Date date)throws ParseException{

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

return sdf.format(date);

}

public static Date parse(String strDate) throws ParseException{

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

return sdf.parse(strDate);

}

}

[IMG_269](http://www.cnblogs.com/peida/archive/2013/05/31/javascript:void(0);)

**说明：**在需要用到SimpleDateFormat 的地方新建一个实例，不管什么时候，将有线程安全问题的对象由共享变为局部私有都能避免多线程问题，不过也加重了创建对象的负担。在一般情况下，这样其实对性能影响比不是很明显的。

**2.使用同步：同步SimpleDateFormat对象**

[IMG_270](http://www.cnblogs.com/peida/archive/2013/05/31/javascript:void(0);)

package com.peidasoft.dateformat;

import java.text.ParseException;import java.text.SimpleDateFormat;import java.util.Date;

public class DateSyncUtil {

private static SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

public static String formatDate(Date date)throws ParseException{

synchronized(sdf){

return sdf.format(date);

}

}

public static Date parse(String strDate) throws ParseException{

synchronized(sdf){

return sdf.parse(strDate);

}

}

}

[IMG_271](http://www.cnblogs.com/peida/archive/2013/05/31/javascript:void(0);)

**说明：**当线程较多时，当一个线程调用该方法时，其他想要调用此方法的线程就要block，多线程并发量大的时候会对性能有一定的影响。

**3.使用ThreadLocal：**

[IMG_272](http://www.cnblogs.com/peida/archive/2013/05/31/javascript:void(0);)

package com.peidasoft.dateformat;

import java.text.DateFormat;import java.text.ParseException;import java.text.SimpleDateFormat;import java.util.Date;

public class ConcurrentDateUtil {

private static ThreadLocal<DateFormat> threadLocal = new ThreadLocal<DateFormat>() {

@Override

protected DateFormat initialValue() {

return new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

}

};

public static Date parse(String dateStr) throws ParseException {

return threadLocal.get().parse(dateStr);

}

public static String format(Date date) {

return threadLocal.get().format(date);

}

}

[IMG_273](http://www.cnblogs.com/peida/archive/2013/05/31/javascript:void(0);)

　　另外一种写法：

[IMG_274](http://www.cnblogs.com/peida/archive/2013/05/31/javascript:void(0);)

package com.peidasoft.dateformat;

import java.text.DateFormat;import java.text.ParseException;import java.text.SimpleDateFormat;import java.util.Date;

public class ThreadLocalDateUtil {

private static final String date\_format = "yyyy-MM-dd HH:mm:ss";

private static ThreadLocal<DateFormat> threadLocal = new ThreadLocal<DateFormat>();

public static DateFormat getDateFormat()

{

DateFormat df = threadLocal.get();

if(df==null){

df = new SimpleDateFormat(date\_format);

threadLocal.set(df);

}

return df;

}

public static String formatDate(Date date) throws ParseException {

return getDateFormat().format(date);

}

public static Date parse(String strDate) throws ParseException {

return getDateFormat().parse(strDate);

}

}

[IMG_275](http://www.cnblogs.com/peida/archive/2013/05/31/javascript:void(0);)

**说明：**使用ThreadLocal, 也是将共享变量变为独享，线程独享肯定能比方法独享在并发环境中能减少不少创建对象的开销。如果对性能要求比较高的情况下，一般推荐使用这种方法。

**4.抛弃JDK，使用其他类库中的时间格式化类：**

1.使用Apache commons 里的FastDateFormat，宣称是既快又线程安全的SimpleDateFormat, 可惜它只能对日期进行format, 不能对日期串进行解析。

2.使用Joda-Time类库来处理时间相关问题

　　做一个简单的压力测试，方法一最慢，方法三最快，但是就算是最慢的方法一性能也不差，一般系统方法一和方法二就可以满足，所以说在这个点很难成为你系统的瓶颈所在。从简单的角度来说，建议使用方法一或者方法二，如果在必要的时候，追求那么一点性能提升的话，可以考虑用方法三，用ThreadLocal做缓存。

　　Joda-Time类库对时间处理方式比较完美，建议使用。

　　参考资料：

　　1.<http://dreamhead.blogbus.com/logs/215637834.html>

　　2.http://www.blogjava.net/killme2008/archive/2011/07/10/354062.html