# [SimpleDateFormat线程不安全及解决办法](http://blog.csdn.net/zdp072/article/details/41044059)

以前没有注意到SimpleDateFormat线程不安全的问题，写时间工具类，一般写成静态的成员变量，不知，此种写法的危险性！在此讨论一下SimpleDateFormat线程不安全问题，以及解决方法。

## 为什么SimpleDateFormat不安全？

直接上一个例子，日期格式化成字符串：

**package** com.anjz.test.simpleDateFormat;

**import** java.text.DateFormat;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**import** java.util.Random;

**import** java.util.concurrent.ExecutorService;

**import** java.util.concurrent.Executors;

/\*\*

\* 测试

\* **@author** ding.shuai

\* **@date** 2017年6月10日下午12:14:19

\*/

**public** **class** SimpleDateFormateTest2 {

**public** **static** **void** main(String[] args) {

**final** DateFormat df = **new** SimpleDateFormat("yyyyMMdd,HHmmss");

ExecutorService ts = Executors.*newFixedThreadPool*(100);

**for** (;;) {

ts.execute(**new** Runnable() {

@Override

**public** **void** run() {

**try** {

//生成随机数，格式化日期

String format = df.format(**new** Date(Math.*abs*(**new** Random().nextLong())));

System.***out***.println(format);

} **catch** (Exception e) {

e.printStackTrace();

System.*exit*(1);

}

}

});

}

}

}

运行一段时间，出现下图的错误



再看一个例子，字符串解析成日期：

**package** com.anjz.test.simpleDateFormat;

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**import** java.util.concurrent.ExecutorService;

**import** java.util.concurrent.Executors;

/\*\*

\* 测试

\* **@author** ding.shuai

\* **@date** 2017年6月10日下午12:38:02

\*/

**public** **class** SimpleDateFormateTest **extends** Thread{

**private** **static** SimpleDateFormat *sdf* = **new** SimpleDateFormat("yyyy-MM-dd");

**private** String name;

**private** String dateStr;

**public** SimpleDateFormateTest(String name,String dateStr) {

**this**.name = name;

**this**.dateStr = dateStr;

}

@Override

**public** **void** run() {

**try** {

Date date = *sdf*.parse(dateStr);

System.***out***.println(name+": date:"+date);

} **catch** (ParseException e) {

e.printStackTrace();

}

}

**public** **static** **void** main(String[] args) {

ExecutorService executorService = Executors.*newFixedThreadPool*(3);

executorService.execute(**new** SimpleDateFormateTest("A", "2017-06-10"));

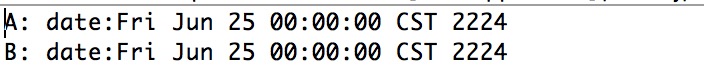
executorService.execute(**new** SimpleDateFormateTest("B", "2016-06-06"));

executorService.shutdown();

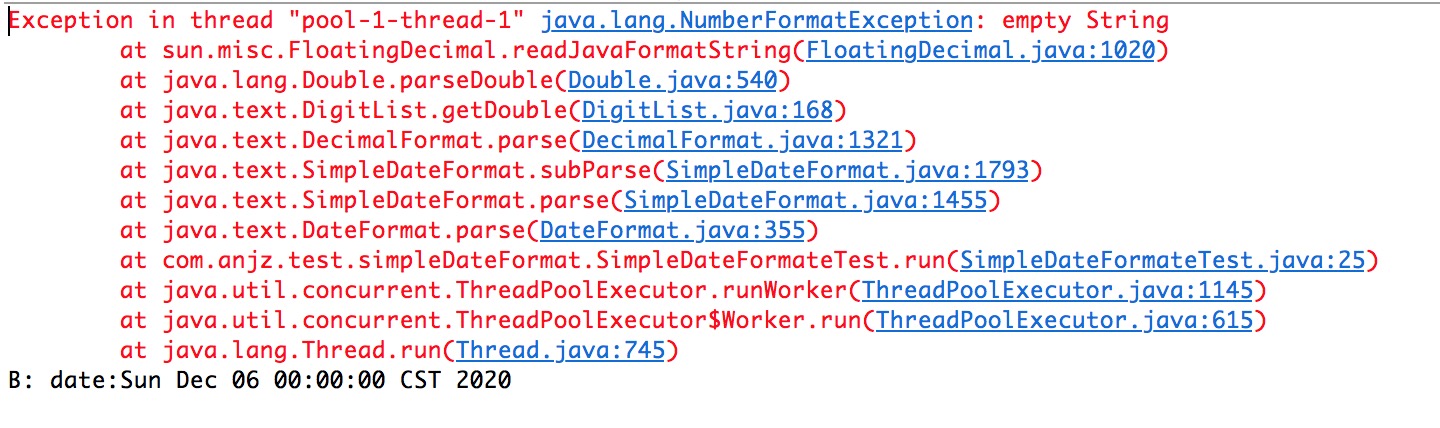
}

}

运行上述例子，可能每次结果都不一样，有时还会报各种错误，如下图，就是笔者运行报的几种错误。







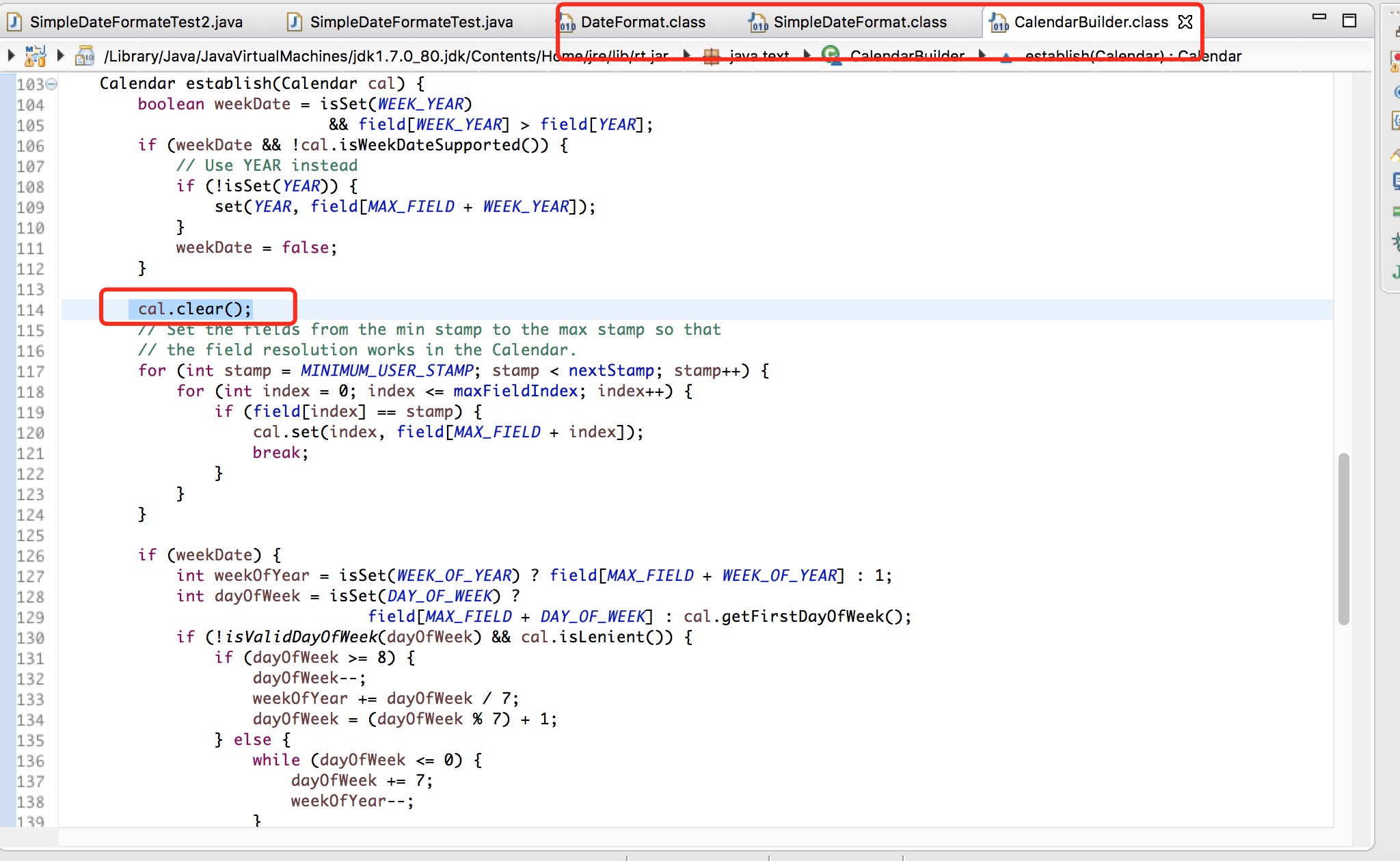
现在我们就分析一下，为什么出现上述原因：

通过查看源码发现，原来SimpleDateFormat类内部有一个Calendar对象引用,它用来储存和这个SimpleDateFormat相关的日期信息,例如sdf.parse(dateStr), sdf.format(date) 诸如此类的方法参数传入的日期相关String, Date等等, 都是交由Calendar引用来储存的.这样就会导致一个问题,**如果你的SimpleDateFormat是个static的, 那么多个thread 之间就会共享这个SimpleDateFormat, 同时也是共享这个Calendar引用**。单例、多线程、又有成员变量（这个变量在方法中是可以修改的），这个场景是不是很像servlet，在高并发的情况下，容易出现幻读成员变量的现象，故说SimpleDateFormat是线程不安全的对象。

ps：servlet因是线程不安全的，所以我们使用servlet的原则是不设置成员变量。

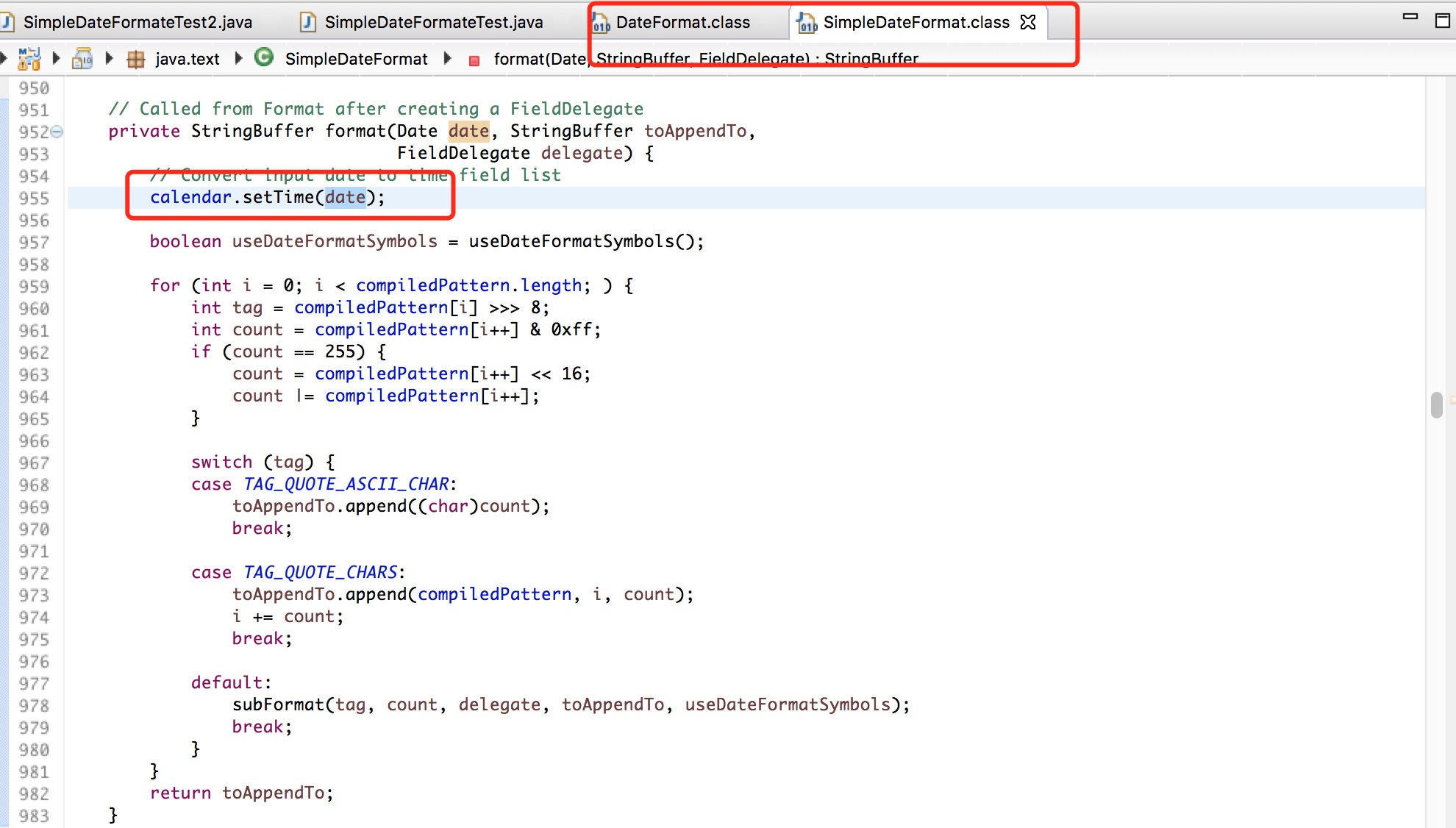
SimpleDateFormat的parse方法：

Calendar是用来承载字符串转化成日期对象的容器，calendar对象有个clear后set值的过程，高并发下，set值的过程，会出现把上次set值给覆盖的情况。



SimpleDateFormat的format方法：

我们传入的日期对象，会直接用Calendar承载，高并发下，Calendar承载的对象也会被覆盖。



## 解决方法

1、将SimpleDateFormat定义成局部变量。

缺点：每调用一次方法就会创建一个SimpleDateFormat对象，方法结束又要作为垃圾回收。

2、方法加同步锁synchronized，在同一时刻，只有一个线程可以执行类中的某个方法。

缺点：性能较差，每次都要等待锁释放后其他线程才能进入。

3、使用第三方库joda-time，由第三方考虑线程不安全的问题。（可以使用）

4、使用ThreadLocal：每个线程拥有自己的SimpleDateFormat对象。（推荐使用）

写一个工具类：

package com.anjz.test.simpleDateFormat;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

/\*\*

\* 时间工具类

\* @author ding.shuai

\* @date 2017年6月10日上午11:31:59

\*/

public class DateUtil {

/\*\*

\* 锁对象

\*/

private static final Object lockObj = new Object();

/\*\*

\* 存放不同的日期模板格式的sdf的Map

\*/

private static Map<String, ThreadLocal<SimpleDateFormat>> sdfMap = new HashMap<String, ThreadLocal<SimpleDateFormat>>();

/\*\*

\* 返回一个ThreadLocal的sdf,每个线程只会new一次sdf

\*

\* @param pattern

\* @return

\*/

private static SimpleDateFormat getSdf(final String pattern) {

ThreadLocal<SimpleDateFormat> tl = sdfMap.get(pattern);

// 此处的双重判断和同步是为了防止sdfMap这个单例被多次put重复的sdf

if (tl == null) {

synchronized (lockObj) {

tl = sdfMap.get(pattern);

if (tl == null) {

// 只有Map中还没有这个pattern的sdf才会生成新的sdf并放入map

System.out.println("put new sdf of pattern " + pattern + " to map");

// 这里是关键,使用ThreadLocal<SimpleDateFormat>替代原来直接new SimpleDateFormat

tl = new ThreadLocal<SimpleDateFormat>() {

@Override

protected SimpleDateFormat initialValue() {

System.out.println("thread: " + Thread.currentThread() + " init pattern: " + pattern);

return new SimpleDateFormat(pattern);

}

};

sdfMap.put(pattern, tl);

}

}

}

return tl.get();

}

/\*\*

\* 使用ThreadLocal<SimpleDateFormat>来获取SimpleDateFormat,这样每个线程只会有一个SimpleDateFormat

\* 如果新的线程中没有SimpleDateFormat，才会new一个

\* @param date

\* @param pattern

\* @return

\*/

public static String format(Date date, String pattern) {

return getSdf(pattern).format(date);

}

public static Date parse(String dateStr, String pattern) throws ParseException {

return getSdf(pattern).parse(dateStr);

}

}

测试代码：

package com.anjz.test.simpleDateFormat;

import java.text.ParseException;

import java.util.Date;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

/\*\*

\* 时间工具类测试

\* @author ding.shuai

\* @date 2017年6月10日上午11:44:01

\*/

public class DateUtilTest {

public static void main(String[] args) throws InterruptedException {

Thread t1 = new Thread(){

@Override

public void run() {

DateUtil.format(new Date(), "yyyy-MM-dd");

};

};

Thread t2 = new Thread(){

@Override

public void run() {

DateUtil.format(new Date(), "yyyy-MM-dd");

};

};

Thread t3 = new Thread(){

@Override

public void run() {

DateUtil.format(new Date(), "yyyy-MM-dd");

};

};

Thread t4 = new Thread(){

@Override

public void run() {

try{

DateUtil.parse("2017-06-10 12:00:01", "yyyy-MM-dd HH:mm:ss");

}catch(ParseException e){

e.printStackTrace();

}

};

};

Thread t5 = new Thread(){

@Override

public void run() {

try{

DateUtil.parse("2017-06-10 12:00:01", "yyyy-MM-dd HH:mm:ss");

}catch(ParseException e){

e.printStackTrace();

}

};

};

System.out.println("单线程执行：");

ExecutorService exec1 = Executors.newFixedThreadPool(1);

exec1.execute(t1);

exec1.execute(t2);

exec1.execute(t3);

exec1.execute(t4);

exec1.execute(t5);

exec1.shutdown();

Thread.sleep(1000);

System.out.println("双线程执行：");

ExecutorService exec2 = Executors.newFixedThreadPool(2);

exec2.execute(t1);

exec2.execute(t2);

exec2.execute(t3);

exec2.execute(t4);

exec2.execute(t5);

exec2.shutdown();

}

}

执行结果：

单线程执行：

put new sdf of pattern yyyy-MM-dd to map

thread: Thread[pool-1-thread-1,5,main] init pattern: yyyy-MM-dd

put new sdf of pattern yyyy-MM-dd HH:mm:ss to map

thread: Thread[pool-1-thread-1,5,main] init pattern: yyyy-MM-dd HH:mm:ss

双线程执行：

thread: Thread[pool-2-thread-1,5,main] init pattern: yyyy-MM-dd

thread: Thread[pool-2-thread-2,5,main] init pattern: yyyy-MM-dd

thread: Thread[pool-2-thread-1,5,main] init pattern: yyyy-MM-dd HH:mm:ss

thread: Thread[pool-2-thread-2,5,main] init pattern: yyyy-MM-dd HH:mm:ss

从结果我们可以看出：

1）1个线程执行5个任务，new了两个SimpleDateFormat对象

2）2个线程执行5个任务，new了四个SimpleDateFormat对象，每个线程拥有两个格式为“yyyy-MM-dd”、“yyyy-MM-dd HH:mm:ss”的对象，线程之间没有共享SimpleDateFormat对象，对于每个线程，都是线性执行，也不会出现共享Calendar的现象。故完美解决SimpleDateFormat线程不安全的问题。

如果使用第一种方式，5个任务的执行，肯定需要new出5个SimpleDateFormat，对于单个线程没有复用的概念。但使用ThreadLocal，对于单个线程，相同格式是可以复用SimpleDateFormat对象，所以最后一种方式，既可以保证线程安全，又可以不耗费系统太多资源，其实这种思想和web中request，response对象是一样的，都是通过线程隔离，每个线程维护一份自己的对象来保证线程安全。

最后附上阿里开发手册中一段关于SimpleDateFormat的规范：

【强制】SimpleDateFormat是线程不安全的类，一般不要定义为static变量，如果定义为static，必须加锁，或者使用DateUtils工具类。

正例：注意线程安全，使用DateUtils。亦推荐如下处理：

private static final ThreadLocal<DateFormat> df = new ThreadLocal<DateFormat>() {

@Override

protected DateFormat initialValue() {

return new SimpleDateFormat("yyyy-MM-dd");

}

};

说明：如果是JDK8的应用，可以使用Instant代替Date，LocalDateTime代替Calendar，DateTimeFormatter代替Simpledateformatter，官方给出的解释：simple beautiful strong immutable thread-safe。