关于 SimpleDateFormat 的线程安全问题

作者：kevin2020
原文链接：https://ld246.com/article/1513618241940
来源网站：链滴

SimpleDateFormat线程安全问题

SimpleDateFormat类内部有一个Calendar对象引用,它用来储存和这个sdf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Calenda引用来储存的

那么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

怎么办呢

头铁一点，并发不高的情况下没什么问题

```
public class DateUtils {
    private static SimpleDateFormat dateFormat=new SimpleDateFormat("yyyy-MM-dd");

    public static String format(Date date){
        return dateFormat.format(date);
    }
}
```

浪费一点，每次都new一个

```
public class DateUtils {
    public static String format(Date date){
        SimpleDateFormat dateFormat=new SimpleDateFormat("yyyy-MM-dd");
        return dateFormat.format(date);
    }
}
```

抠门一点，大家排队共用一个

```
public class DateUtils {
    private static SimpleDateFormat dateFormat=new SimpleDateFormat("yyyy-MM-dd");

    public static synchronized String format(Date date){
        return dateFormat.format(date);
    }
}
```

稳健一点，每个单位分一个，单位内有序使用 很棒

```
public class DateUtilsDateLocal {
    private static ThreadLocal<DateFormat> threadLocal=new ThreadLocal<DateFormat>(){
        @Override
        protected  DateFormat initialValue(){
            return new SimpleDateFormat("yyyy-MM-dd");
        }
    };
    public static String format(Date date){
        return threadLocal.get().format(date);
    }
}
```

或者抛弃JDK，使用其他类库中的时间格式化类：

原文链接：关于 SimpleDateFormat 的线程安全问题

1.使用Apache commons 里的FastDateFormat，宣称是既快又线程安全的SimpleDateFormat, 可它只能对日期进行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

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

既然上面引出了ThrealLocal 下面简单介绍一下

ThreadLocal作用及使用方式

ThreadLocal并不能解决同一变量的共享问题，它只是为每个线程维护了线程私有对象的拷贝，而在个线程内，可以看作是单线程的，就不会引起并发的问题

打个比方 4个班级100个学生 都要看漫画，而ThreadLocal就好比为4个班级各发一本漫画，因为班级有老师的存在，不存在争抢的问题，每个学生都只能排队看漫画，这样就不会带来争抢的问题。

ThreadLocal的构造函数签名是这样的：

```
    /**
     * Creates a thread local variable.
     * @see #withInitial(java.util.function.Supplier)
     */
    public ThreadLocal() {
    }
```

内部啥也没做。

initialValue函数

initialValue函数用来设置ThreadLocal的初始值，函数签名如下：

```
    protected T initialValue() {
        return null;
    }
```

该函数在调用get函数的时候会第一次调用，但是如果一开始就调用了set函数，则该函数不会被调用通常该函数只会被调用一次，除非手动调用了remove函数之后又调用get函数，这种情况下，get函中还是会调用initialValue函数。该函数是protected类型的，很显然是建议在子类重载该函数的，所通常该函数都会以匿名内部类的形式被重载，以指定初始值，比如：

```
public class TestThreadLocal {
    private static final ThreadLocal<Integer> value = new ThreadLocal<Integer>() {
        @Override
        protected Integer initialValue() {
            return Integer.valueOf(1);
        }
    };
}
```

get函数

该函数用来获取与当前线程关联的ThreadLocal的值，函数签名如下：

```
public T get()
```

如果当前线程没有该ThreadLocal的值，则调用initialValue函数获取初始值返回。

set函数

set函数用来设置当前线程的该ThreadLocal的值，函数签名如下：

```
public void set(T value)
```

设置当前线程的ThreadLocal的值为value。

remove函数

remove函数用来将当前线程的ThreadLocal绑定的值删除，函数签名如下：

```
public void remove()
```

在某些情况下需要手动调用该函数，防止内存泄露。