



(<https://www.aliyun.com/minisite/goods?userCode=y25lwhrv>)

ThreadLocal图解

发布时间:2019/09/06| 阅读: 111

前言

到底什么是线程的不安全？为什么会存在线程的不安全？线程的不安全其实就是多个线程并发的去操作同一共享变量没用做同步所产生意料之外的结果。那是如何体现出来的呢？我们看下面的一个非常经典的例子:两个操作员同时操作同一个银行账户，A操作员存钱，100B操作员取钱50。我们看一下流程。

A操作员存100元



B操作员取50元

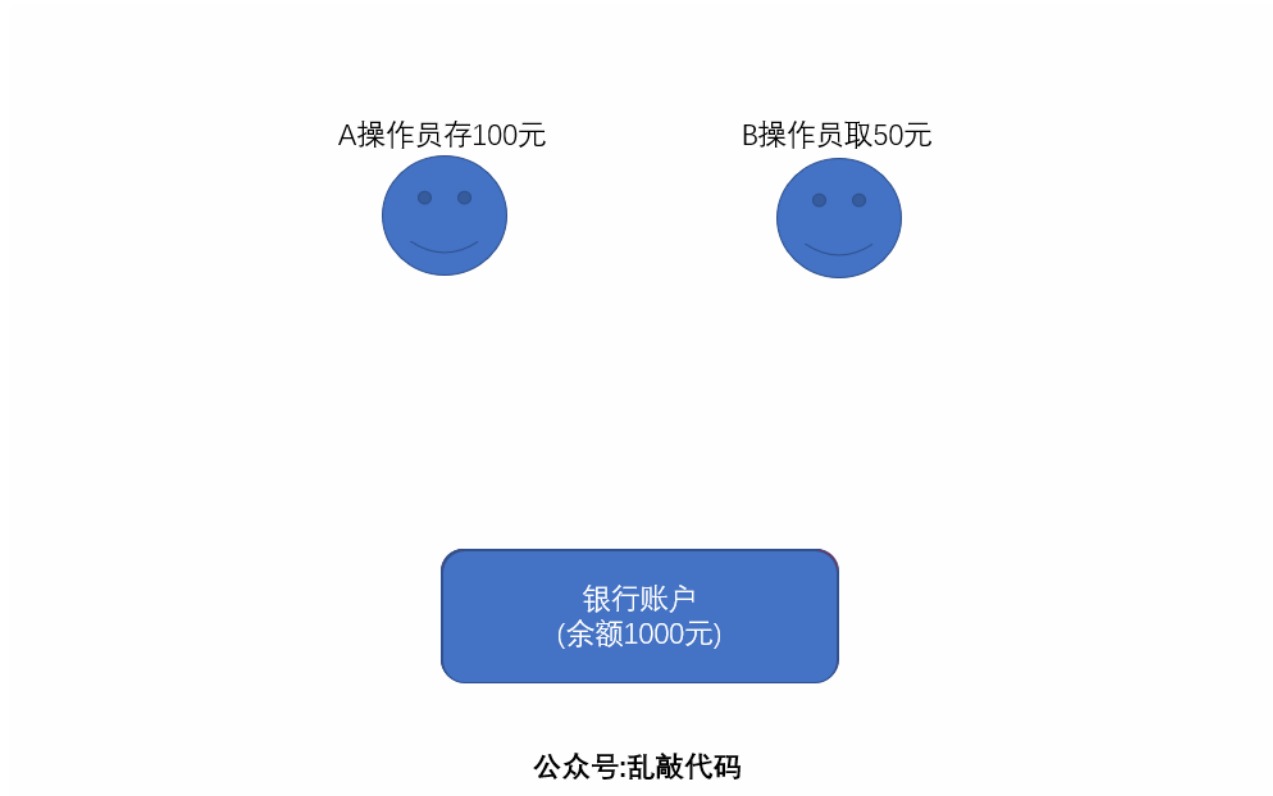


银行账户
(余额1000元)

公众号:乱敲代码

两个操作员同时处理，没用做同步这个时候我们发现银行账户最终余额剩余950元，在我们想的最终结果银行账户应该剩余 $1000+100-50=1050$ 元，在执行过程中我们没有加锁，最终导致了运行结果偏离预期。那么如何解决的？一般的解决措施就是加锁，加同步锁所以这就需要使用者一定要知道锁是什么。

我们来看一下加锁之后的效果是不是我们所预期的。



在添加同步锁后我们可以看到，A操作员和B操作员同时去操作账户，但是A先抢占到资源，所以B就只能等待A操作员释放锁才能去操作银行账户，那么最终结果是我们所预期的吗？答案是的。

同步的话一般都是加锁，如果现在我想创建多个线程每个线程都是访问的自己的变量呢？各个线程之间毫无关联？

答案是有的。

ThreadLocal问题

ThreadLocal 是JDK提供的，它提供了线程本地变量。什么是线程本地变量呢？其实就是你创建了一个 Threadlocal 变量，每个访问 Threadlocal 变量的线程都有一个本地副本。我们看下面的图：



公众号:乱敲代码

从上面看出你创建一个 `ThreadLocal` 变量,每个访问该的线程都会复制到自己的本地,所以线程操作的都是本地的副本,这也就是说每个线程都是操作的自己本地的变量,那就完美的避免了线程安全的问题。

在这里还有一个问题。我在写这篇文章的时候看过很多文章,总的来说就是 `ThreadLocal` 就是为了解决多线程并发问题而提供的一种方法,还有一种解释就是 `ThreadLocal` 的最终目的就是为了解决多线程访问共享资源所产生的。真的对吗? `ThreadLocal` 并没有共享那么从何而来的同步呢?

自己的想法

在看了 `Java`并发编程之美 后我所理解的 `ThreadLocal` 提供了线程本地变量的副本,每个线程实际操作的是自己本地的变量副本,也就是说该变量副本只能当前线程访问,就不存在多个线程共享的问题,从 `ThreadLocal` 名字我们也能看出 本地线程。那它那它也就不存在去解决并发问题了。

如何使用

我们来看下面的例子。

```

static ThreadLocal<Integer> threadLocal = new ThreadLocal<Integer>();
public static void main(String[] args) {
    new Thread()->{
        threadLocal.set(new Random().nextInt(100));
        System.out.println(Thread.currentThread()+"===="+threadLocal.get());
    }.start();
    new Thread()->{
        threadLocal.set(new Random().nextInt(100));
        System.out.println(Thread.currentThread()+"===="+threadLocal.get());
    }.start();
}

```

输出结果:

Thread[Thread-1,5,main]====57

Thread[Thread-0,5,main]====75

创建了两个线程,它们都在 `threadLocal` 上面都set了一个随机数,我们看最后得输出结果每个都是不同得值,那么我们如果把 `threadLocal` 替换成一个集合会发生什么,由于两个线程时上个线程生成的随机数57会被第二个线程覆盖掉,而在 `Threadlocal` 中两个线程都是操作的自己的本地副本,那么两个线程互不影响都无法操控到对方的数据,因此它们存取的都是不同的值。

实现原理

那么 `Threadlocal` 是如何实现的呢? 在研究 `Threadlocal` 的实现原理我们先看一下 `Thread` 的内部属性。

```

/* ThreadLocal values pertaining to this thread. This map is maintained
 * by the ThreadLocal class. */
ThreadLocal.ThreadLocalMap threadLocals = null;

/*
 * InheritableThreadLocal values pertaining to this thread. This map is
 * maintained by the InheritableThreadLocal class.
 */
ThreadLocal.ThreadLocalMap inheritableThreadLocals = null;

```

- `threadLocals` 此线程保存的 `Threadlocal` 的值

`inheritableThreadLocals`等到后面再说。


在 `Thread` 的内部属性中我们看到了这两个默认为`null`的属性，`threadLocals`用来保存 `Threadlocal` 的本地副本，默认是为`null`只有调用 `Threadlocal` 的`set`时才会创建。也就是说 `Threadlocal` 就类似一个工具，它的作用就是把`value`的值通过`set`存在线程每个线程的`threadLocals` 中，只要线程一直存在 `threadLocals` 也就一直存在。所以当不需要使用本地变量的时候可以调用 `Threadlocal` 的`remove`来清空本地变量。而`threadLocals` 为什么继承鱼`ThreadLocalMap`呢?`ThreadLocalMap`是一个定制的 `HashMap`，而使用`Map`的原因就是可以每个线程关联多个 `Threadlocal` 变量。

set方法

我们来看一下`set`方法是如何实现的。


```
public void set(T value) {  
    //首先获取当前线程  
    Thread t = Thread.currentThread();  
    //通过当前线程去获取对应的线程变量  
    ThreadLocal.ThreadLocalMap map = getMap(t);  
    if (map != null)  
        //找到了就set进去key使用当前实例  
        map.set(this, value);  
    else  
        //没有找到就创建一个  
        createMap(t, value);  
}
```

可以看出流程非常简单，首先获取当前线程然后在进行下一步操作，我们在看一下`getMap`做了什么



```
ThreadLocal.ThreadLocalMap getMap(Thread t) {  
    //返回当前线程的threadLocals  
    return t.threadLocals;  
}
```

getMap主要就是返回了当前threadLocals的属性。那如果map为空呢？



```
void createMap(Thread t, T firstValue) {  
    t.threadLocals = new ThreadLocalMap(this, firstValue);  
}
```

如果map为空的话就直接创建一个新的ThreadLocalMap。

我们来看一下流程图。

get方法

看一下Get方法

```
public T get() {
    //获取当前线程
    Thread t = Thread.currentThread();
    //通过当前线程获取对应的线程变量
    ThreadLocal.ThreadLocalMap map = getMap(t);
    if (map != null) {
        //不等于空就通过当前实例获取属性
        ThreadLocal.ThreadLocalMap.Entry e = map.getEntry(this);
        //属性等于空就直接返回
        if (e != null) {
            @SuppressWarnings("unchecked")
            T result = (T)e.value;
            return result;
        }
    }
    //返回空
    return setInitialValue();
}

private T setInitialValue() {
    //初始化一个空
    T value = initialValue();
    //获取当前实例
    Thread t = Thread.currentThread();
    //通过当前线程获取对应的线程变量
    ThreadLocal.ThreadLocalMap map = getMap(t);
    if (map != null)
        //不等于空就set
        map.set(this, value);
    else
        //等于空就创建
        createMap(t, value);
    //返回属性
    return value;
}
```

首先根据当前线程获取实例如果存在就返回，如果不存在就先初始化一个空值，然后判断如果当前threadLocals不为空就直接set一个空，否则就创建一个变量。

remove方法

```

public void remove() {
    //先获取线程实例，然后根据当前线程获取threadlocals变量
    ThreadLocal.ThreadLocalMap m = getMap(Thread.currentThread());
    //threadlocals变量不为空就删除
    if (m != null)
        m.remove(this);
}

```

remove方法相对来说比较简单。

总结

Threadlocal 的实现原理其实就是通过set把value set到线程的threadlocals属性中， threadlocals 类型是Map其中的Key就是 Threadlocal 的this引用，value就是我们所set的值，如果当前线程不销毁的话 threadlocals 会一直存在。一直存在的话可能会造成内存溢出，所以使用完之后尽量remove一下。不过在这里又有一个问题那就是如果我的线程想要读取主线程的变量要怎么做？我们上面的例子都是设置的新创建的线程，那么现在我在主线程中set一个值，这个时候我在新创建的线程中可以读取到吗？答案是不可以，因为 Threadlocal 不支持继承性。

我们看下面的例子：

```

static ThreadLocal<Integer> threadLocal = new ThreadLocal<Integer>();
public static void main(String[] args) {
    threadLocal.set(1000);
    new Thread()->{
        System.out.println(Thread.currentThread()+"===="+threadLocal.get());
    }.start();
}

```

输出结果：

Thread[Thread-0,5,main]====null

也就是说 `ThreadLocal` 不支持继承性，主线程设置了值，在子线程中是获取不到的。那我现在想要获取主线程里面的值要怎么做？

`ThreadLocal` 是实现不了的，不过 `ThreadLocal` 有一个子类可以实现。`InheritableThreadLocal`，`InheritableThreadLocal` 是 `ThreadLocal` 的实现，我们来看一个简单的例子。

```
static ThreadLocal<Integer> threadLocal = new InheritableThreadLocal<Integer>();
public static void main(String[] args) {
    threadLocal.set(1000);
    new Thread(()->{
        System.out.println(Thread.currentThread()+ "===="+threadLocal.get());
    }).start();
}
```

输出结果：

`Thread[Thread-0,5,main]====1000`

运行结果发现子线程是可以获取到主线程设置的值的，那它是如何实现的？

我们看一下代码实现：

```
void createMap(Thread t, T firstValue) {
    t.inheritableThreadLocals = new ThreadLocal.ThreadLocalMap(this, firstValue);
}
ThreadLocal.ThreadLocalMap getMap(Thread t) {
    return t.inheritableThreadLocals;
}
protected T childValue(T parentValue) {
    return parentValue;
}
```

`InheritableThreadLocal` 是继承 `ThreadLocal` 的，并且把 `threadLocals` 给替换成 `inheritableThreadLocals` 了所以上面的 `inheritableThreadLocals` 我要留在最后说，那么替换成 `inheritableThreadLocals` 后子线程就可以获取到主线程设置的属性了吗？我们在看一下 `Thread` 的实现。

```
private void init(ThreadGroup g, Runnable target, String name,
                  long stackSize, AccessControlContext acc,
                  boolean inheritThreadLocals) {
    if (name == null) {
        throw new NullPointerException("name cannot be null");
    }

    this.name = name;
    //获取当前变量
    Thread parent = currentThread();
    //如果主线程的inheritableThreadLocals不等于空
    if (inheritThreadLocals && parent.inheritableThreadLocals != null)
        //设置inheritableThreadLocals
        this.inheritableThreadLocals =
            ThreadLocal.createInheritedMap(parent.inheritableThreadLocals);
    /* Stash the specified stack size in case the VM cares */
    this.stackSize = stackSize;

    /* Set thread ID */
    tid = nextThreadID();
}
```

看 Thread 的初始化方法可以看出，先获取了当前线程(主线程)判断主线程的 `inheritableThreadLocals` 不为空的话就调用`createInheritedMap`方法赋值给子线程中的 `inheritableThreadLocals` 。具体这里解释太多。有机会在写一篇文章来解释。

本文完

本文作者：孙罗蒙

本文链接：p/39aaf51b.html ([/p/39aaf51b.html](http://p/39aaf51b.html))

版权： 本站文章均采用 CC BY-NC-SA 3.0 CN (<http://creativecommons.org/licenses/by-nc-sa/3.0/cn/>) 许可协议，请勿用于商业，转载注明出处！

相关文章

[HashMap源码分析\(二\):看完彻底了解HashMap \(\p3101897f.html\)](http://p3101897f.html)

[HashSet源码分析:JDK源码系列 \(\p7a26c329.html\)](http://p7a26c329.html)

[JAVA内存模型详解\(一\) \(\p95408866.html\)](http://p95408866.html)

[使用Arrays工具类操作数组 \(\pa7c03e49.html\)](http://pa7c03e49.html)

[先看Java内存模型在看并发 \(\p135fb432.html\)](http://p135fb432.html)

昵称

邮箱

网址(http://)

说点什么



表情 | 预览



(<https://segmentfault.com/markdown>)

回复

2 评论



乱敲代码 (lqcoder.com) Safari 4.0 Android 9

2019-12-01

回复

主题又自己开发了



乱敲代码 (sunluomeng.top) Chrome 77.0.3865.90 Windows 10.0

2019-11-07

回复

主题已经更新为Next 😊 😎

Powered By Valine (<https://valine.js.org>)

v1.3.10

已运行166 天 22 小时 38 分钟 36 秒

@孙罗蒙

本站访客数:4371 | 本站访问量9674

豫ICP备19020769号 (<http://www.beian.miit.gov.cn/>)