一、线程本地变量ThreadLocal

1. 基本概念

(1) 线程本地变量是说，每个线程都有同一个变量的独有拷贝。只有该类变量才具有的特性。

2. 类TheadLocal的用法

2.1 主要方法

(1) ThreadLocal是一个泛型类，接受一个类型参数T，它只有一个空的构造方法，有两个主要的public方法：

public T get() //获取值，如果没有返回null

public void set(T value) //设置值

(2) 实例代码：

public class ThreadLocalBasic {

//静态变量local

static ThreadLocal<Integer> local = new ThreadLocal<>();

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

Thread child = new Thread() {

@Override

public void run() {

//在线程体重设置local为200

System.out.println("child thread initial: " + local.get());

local.set(200);

System.out.println("child thread final: " + local.get());

}

};

//在主线程中设置local为100

local.set(100);

child.start();

child.join();

System.out.println("main thread final: " + local.get());

}

}

① 输出：

child thread initial: null

child thread final: 200

main thread final: 100

② 解析：

main线程对local变量的设置对child线程不起作用，child线程对local变量的改变也不会影响main线程，它们访问的虽然是同一个变量local，但每个线程都有自己的独立的值，这就是线程本地变量的含义。

(3) 其他方法

protected T initialValue()

public void remove()

① 注意：

initialValue用于提供初始值，它是一个受保护方法，可以通过匿名内部类的方式提供，当调用get方法时，如果之前没有设置过，会调用该方法获取初始值，默认实现是返回null。remove删掉当前线程对应的值，如果删掉后，再次调用get，会再调用initialValue获取初始值。

3. ThreadLocal应用场景 -- ThreadLocal是实现线程安全的一种方案

3.1 DateFormat/SimpleDateFormat

(1) 实例代码：

public class ThreadLocalDateFormat {

//新建一个线程本地变量，并且重写初始化方法

static ThreadLocal<DateFormat> sdf = new ThreadLocal<DateFormat>() {

@Override

protected DateFormat initialValue() {

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

}

};

public static String date2String(Date date) {

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

}

public static Date string2Date(String str) throws ParseException {

return sdf.get().parse(str);

}

}

(2) 需要说明的是，ThreadLocal对象一般都定义为static，以便于引用。

3.2 ThreadLocalRandom

(1) Random是线程安全的，但如果并发访问竞争激烈的话，性能会下降，所以Java并发包提供了类ThreadLocalRandom，它是Random的子类，利用了ThreadLocal，它没有public的构造方法，通过静态方法current获取对象。

public static void main(String[] args) {

ThreadLocalRandom rnd = ThreadLocalRandom.current();

System.out.println(rnd.nextInt());

}

3.3 上下文信息

(1) 背景

ThreadLocal的典型用途是提供上下文信息，比如在一个Web服务器中，一个线程执行用户的请求，在执行过程中，很多代码都会访问一些共同的信息，比如请求信息、用户身份信息、数据库连接、当前事务等，它们是线程执行过程中的全局信息，如果作为参数在不同代码间传递，代码会很啰嗦，这时，使用ThreadLocal就很方便，所以它被用于各种框架如Spring中。

(2) 实例代码：

public class RequestContext {

public static class Request { //...

};

//将要被并发访问的变量设为线程本地变量

private static ThreadLocal<String> localUserId = new ThreadLocal<>();

private static ThreadLocal<Request> localRequest = new ThreadLocal<>();

public static String getCurrentUserId() {

return localUserId.get();

}

public static void setCurrentUserId(String userId) {

localUserId.set(userId);

}

public static Request getCurrentRequest() {

return localRequest.get();

}

public static void setCurrentRequest(Request request) {

localRequest.set(request);

}

}

4. 基本原理

(1) 每个线程都有一个Map，类型为ThreadLocalMap，调用set实际上是在线程自己的Map里设置了一个条目，键为当前的ThreadLocal对象，值为value。ThreadLocalMap是一个内部类，它是专门用于ThreadLocal的，与一般的Map不同，它的键类型为WeakReference<ThreadLocal>，我们没有提过WeakReference，它与Java的垃圾回收机制有关，使用它，便于回收内存。

(2) 每个线程都有一个Map，对于每个ThreadLocal对象，调用其get/set实际上就是以ThreadLocal对象为键读写当前线程的Map，这样，就实现了每个线程都有自己的独立拷贝的效果。

二、线程池与ThreadLocal

(1) 问题

线程池中的线程在执行完一个任务，执行下一个任务时，其中的ThreadLocal对象并不会被清空，修改后的值带到了下一个异步任务。因为线程池里的线程是复用的，可以是一个线程先后处理多个任务。

(2) 解决方法

① 第一次使用ThreadLocal对象时，总是先调用set设置初始值，或者如果ThreaLocal重写了initialValue方法，先调用remove

② 使用完ThreadLocal对象后，总是调用其remove方法

③ 使用自定义的线程池