链滴

复现一个典型的线上 Spring Bean 对象的线程安全问题（附三种解决办法）

作者：liumapp

原文链接：https://ld246.com/article/1563263041190

来源网站：链滴

问题复现

假设线上是一个典型的Spring Boot Web项目，某一块业务的处理逻辑为：

接受一个name字符串参数，然后将该值赋予给一个注入的bean对象，修改bean对象的name属性后返回，期间我们用了 Thread.sleep(300) 来模拟线上的高耗时业务

代码如下：

```
@RestController
@RequestMapping("name")
public class NameController {

    @Autowired
    private NameService nameService;

    @RequestMapping("")
    public String changeAndReadName (@RequestParam String name) throws InterruptedExce
tion {
        System.out.println("get new request: " + name);
        nameService.setName(name);
        Thread.sleep(300);
        return nameService.getName();
    }

}
```

上述的nameService也非常简单，一个普通的Spring Service对象

具体代码如下所示：

```
@Service
public class NameService {

    private String name;

    public NameService() {
    }

    public NameService(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public NameService setName(String name) {
        this.name = name;
        return this;
    }
}
```

相信使用过Spring Boot的伙伴们对这段代码不会有什么疑问，实际运行也没有问题，测试也能跑通但真的上线后，里面却会产生一个线程安全问题

不相信的话，我们通过线程池，开200个线程来测试NameController就可以复现出来

测试代码如下

```
    @Test
    public void changeAndReadName() throws Exception {
        ThreadPoolExecutor poolExecutor = new ThreadPoolExecutor(200, 300 , 2000, TimeUnit.
ECONDS, new ArrayBlockingQueue<Runnable>(200));
        for (int i = 0; i < 200; i++) {
            poolExecutor.execute(new Runnable() {
                @Override
                public void run() {
                    try {
                        System.out.println(Thread.currentThread().getName() + " begin");
                        Map<String, String> headers = new HashMap<String, String>();
                        Map<String, String> querys = new HashMap<String, String>();

                        querys.put("name", Thread.currentThread().getName());
                        headers.put("Content-Type", "text/plain;charset=UTF-8");
                        HttpResponse response = HttpTool.doGet("http://localhost:8080",
                                "/name",
                                "GET",
                                headers,
                                querys);
                        String res = EntityUtils.toString(response.getEntity());

                        if (!Thread.currentThread().getName().equals(res)) {
                            System.out.println("WE FIND BUG !!!");
                            Assert.assertEquals(true, false);
                        } else {
                            System.out.println(Thread.currentThread().getName() + " get received " + re
);
                        }
                    }catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            });
        }
        while(true) {
            Thread.sleep(100);
        }
    }
```

这段测试代码，启动200个线程，对NameController进行测试，每一个线程将自己的线程名作为参提交，并对返回结果进行断言，如果返回的值与提交的值不匹配，那么抛出AssertNotEquals异常

实际测试后，我们可以发现200个线程近乎一半以上都会抛出异常

问题产生原因

首先我们来分析一下，当一个线程，向 http://localhost:8080/name 发出请求时，线上的Spring Boo服务，会通过其内置的Tomcat 8.5来接收这个请求

而在Tomcat 8.5中，默认采用的是NIO的实现方式，及每次请求对应一个服务端线程，然后这个服务的线程，再分配到对应的servlet来处理请求

所以我们可以认为，这并发的200次客户端请求，进入NameController执行请求的，也是分为200个同的服务端线程来处理

但是Spring提供的Bean对象，并没有默认实现它的线程安全性，即默认状态下，我们的NameControler跟NameService都属于单例对象

这下应该很好解释了，200个线程同时操作2个单例对象（一个NameController对象，一个NameServce对象），在没有采用任何锁机制的情况下，不产生线程安全问题是不可能的（除非是状态无关性操）

问题解决办法

按照标题说明的，我这里提供三种解决办法，分别是

- synchronized修饰方法
- synchronized代码块
- 改变bean对象的作用域

接下来对每个解决办法进行说明，包括他们各自的优缺点

synchronized修饰方法

使用synchronized来是修饰可能会产生线程安全问题的方法，应该是我们最容易想到的，同时也是最单的解决办法，我们仅仅需要在 public String changeAndReadName (@RequestParam String name) 这个方法上，增加一个synchronized进行修饰即可

实际测试，这样确实能解决问题，但是各位是否可以再思考一个问题

我们再来运行测试代码的时候，发现程序运行效率大大降低，因为每一个线程必须等待前一个线程完changeAndReadName()方法的所有逻辑后才可以运行，而这段逻辑中，就包含了我们用来模拟高时业务的 Thread.sleep(300)，但它跟我们的线程安全没有什么关系

这种情况下，我们就可以使用第二种方法来解决问题

synchronized代码块

实际的线上逻辑，经常会遇到这样的情况：我们需要确保线程安全的代码，跟高耗时的代码(比如说调第三方api)，很不凑巧的写在同一个方法中

那么这种情况下，使用synchronized代码块，而不是直接修饰方法会来得高效的多

具体解决代码如下：

```
    @RequestMapping("")
    public String changeAndReadName (@RequestParam String name) throws InterruptedExce
tion {
```

```
        System.out.println(Thread.currentThread().getName() + " get new request: " + name);
        String result = "";
        synchronized (this) {
            nameService.setName(name);
            result = nameService.getName();
        }
        Thread.sleep(300);
        return result;
    }
```

再次运行测试代码，我们可以发现效率问题基本解决，但是缺点是需要我们自己把握好哪一块是可能现线程安全问题的代码（而实际的线上逻辑可能非常复杂，这一块不好把握）

改变bean对象的作用域

现在非常不幸的事情发生了，我们连高耗时代码也是状态相关性的，而同时也需要保证效率问题，那这种情况下就只能通过牺牲少量的内存来解决问题了

大概思路就是通过改变bean对象的作用域，让每一个服务端线程对应一个新的bean对象来处理逻辑通过彼此之间互不相关来回避线程安全问题

首先我们需要知道bean对象的作用域有哪些，请见下表

作用域	说明
singleton	默认的作用域，这种情况下的bean都被定义为一个单例对象，该对象的生命周期是与Spring IOC容器一致的（但出于Spring懒加载机制，有在第一次被使用时才会创建）
prototype	bean被定义为在每次注入时都会创建个新的对象
request	bean被定义为在每个HTTP请求中创建个单例对象，也就是说在单个请求中都会复用这一个单例对象
session	bean被定义为在一个session的生命周期创建一个单例对象
application	bean被定义为在ServletContext的命周期中复用一个单例对象
websocket	bean被定义为在websocket的生命期中复用一个单例对象

清楚bean对象的作用域后，接下来我们就只需要考虑一个问题：修改哪些bean的作用域?

前面我已经解释过，这个案例中，200个服务端线程，在默认情况下是操作2个单例bean对象，分别NameController和NameService（没错，在Spring Boot下，Controller默认也是单例对象）

那么是不是直接将NameController和NameServie设置为prototype就可以了呢?

如果您的项目是用的Struts2，那么这样做没有任何问题，但是在Spring MVC下会严重影响性能，因Struts2对请求的拦截是基于类，而Spring MVC则是基于方法

所以我们应该将NameController的作用域设置为request，将NameService设置为prototype来解决

具体操作代码如下

```
@RestController
@RequestMapping("name")
@Scope("request")
public class NameController {

}

@Service
@Scope("prototype")
public class NameService {

}
```

参考文献

- https://dzone.com/articles/understanding-spring-reactiveclient-to-server-comm
- https://dzone.com/articles/understanding-spring-reactive-servlet-async
- https://medium.com/sipios/how-to-make-parallel-calls-in-java-springboot-application-and how-to-test-them-dcc27318a0cf

案例项目代码： github/liumapp/booklet