链滴

面试：Spring 中的 bean 是线程安全的吗?

作者：jianzh5

原文链接：https://ld246.com/article/1603423851866

来源网站：链滴

面试官经常喜欢问Spring中的bean是不是线程安全的这个问题用来考察对Spring中Bean作用域的理，先说结论，Spring中的Bean不是线程安全的。

Spring容器中的Bean是否线程安全，容器本身并没有提供Bean的线程安全策略，因此可以说Spring器中的Bean本身不具备线程安全的特性，但是具体还是要结合具体scope的Bean去研究。

Spring Bean作用域

Spring 的 bean 作用域（scope）类型

1、singleton:单例，默认作用域。

2、prototype:原型，每次创建一个新对象。

3、request:请求，每次Http请求创建一个新对象，适用于WebApplicationContext环境下。

4、session:会话，同一个会话共享一个实例，不同会话使用不用的实例。

5、global-session:全局会话，所有会话共享一个实例。

线程安全这个问题，要从单例与原型Bean分别进行说明。

原型Bean

对于原型Bean,每次创建一个新对象，也就是线程之间并不存在Bean共享，自然是不会有线程安全的题。

单例Bean

对于单例Bean,所有线程都共享一个单例实例Bean,因此是存在资源的竞争。

如果单例Bean,是一个无状态Bean，也就是线程中的操作不会对Bean的成员执行**查询**以外的操作，那这个单例Bean是线程安全的。比如Spring mvc 的 Controller、Service、Dao等，这些Bean大多是状态的，只关注于方法本身。

spring单例，为什么controller、service和dao确能保证线程安全?

Spring中的Bean默认是单例模式的，框架并没有对bean进行多线程的封装处理。实际上大部分时间Ban是无状态的（比如Dao） 所以说在某种程度上来说Bean其实是安全的。

但是如果Bean是有状态的 那就需要开发人员自己来进行线程安全的保证，最简单的办法就是改变bea的作用域 把 singleton 改为 protopyte， 这样每次请求Bean就相当于是 new Bean() 这样就可以保线程的安全了。

有状态就是有数据存储功能

无状态就是不会保存数据

controller、service和dao层本身并不是线程安全的，只是如果只是调用里面的方法，而且多线程调一个实例的方法，会在内存中复制变量，这是自己的线程的工作内存，是安全的。

想理解原理可以看看《深入理解JVM虚拟机》，2.2.2节：

Java虚拟机栈是线程私有的，它的生命周期与线程相同。虚拟机栈描述的是Java方法执行的内存模型

原文链接：面试：Spring 中的 bean 是线程安全的吗?

每个方法在执行的同时都会创建一个栈帧用于存储局部变量表、操作数栈、动态链接、方法出口等信息。

《Java并发编程实战》第3.2.2节：

局部变量的固有属性之一就是封闭在执行线程中。
它们位于执行线程的栈中，其他线程无法访问这个栈。

所以其实任何无状态单例都是线程安全的。Spring的根本就是通过大量这种单例构建起系统，以事务本的方式提供服务

也可以看看这篇加深理解： 关于Spring的@Controller @Service等的线程安全问题

首先问@Controller @Service是不是线程安全的?

答：默认配置下不是的。为啥呢？因为默认情况下@Controller没有加上@Scope，没有加@Scope是默认值singleton，单例的。意思就是系统只会初始化一次Controller容器，所以每次请求的都是同个Controller容器，当然是非线程安全的。举个栗子：

```
@RestController
public class TestController {

    private int var = 0;

    @GetMapping(value = "/test_var")
    public String test() {
        System.out.println("普通变量var:" + (++var));
        return "普通变量var:" + var ;
    }
}
```

在postman里面发三次请求，结果如下：

```
普通变量var:1
普通变量var:2
普通变量var:3
```

说明他不是线程安全的。怎么办呢？可以给他加上上面说的@Scope注解，如下：

```
@RestController
@Scope(value = "prototype") // 加上@Scope注解，他有2个取值：单例-singleton 多实例-prototype
public class TestController {

    private int var = 0;

    @GetMapping(value = "/test_var")
    public String test() {
        System.out.println("普通变量var:" + (++var));
        return "普通变量var:" + var ;
    }
}
```

这样一来，每个请求都单独创建一个Controller容器，所以各个请求之间是线程安全的，三次请求结：

```
普通变量var:1
普通变量var:1
普通变量var:1
```

加了@Scope注解多的实例prototype是不是一定就是线程安全的呢?

```
@RestController
@Scope(value = "prototype") // 加上@Scope注解，他有2个取值：单例-singleton 多实例-prototype
public class TestController {
    private int var = 0;
    private static int staticVar = 0;

    @GetMapping(value = "/test_var")
    public String test() {
        System.out.println("普通变量var:" + (++var)+ "---静态变量staticVar:" + (++staticVar));
        return "普通变量var:" + var + "静态变量staticVar:" + staticVar;
    }
}
```

看三次请求结果:

```
普通变量var:1---静态变量staticVar:1
普通变量var:1---静态变量staticVar:2
普通变量var:1---静态变量staticVar:3
```

虽然每次都是单独创建一个Controller但是扛不住他变量本身是static的呀，所以说呢，即便是加上@cope注解也不一定能保证Controller 100%的线程安全。所以是否线程安全在于怎样去定义变量以及ontroller的配置。所以来个全乎一点的实验，代码如下:

```
@RestController
@Scope(value = "singleton") // prototype singleton
public class TestController {

    private int var = 0; // 定义一个普通变量

    private static int staticVar = 0; // 定义一个静态变量

    @Value("${test-int}")
    private int testInt; // 从配置文件中读取变量

    ThreadLocal<Integer> tl = new ThreadLocal<>(); // 用ThreadLocal来封装变量

    @Autowired
    private User user; // 注入一个对象来封装变量

    @GetMapping(value = "/test_var")
    public String test() {
        tl.set(1);
        System.out.println("先取一下user对象中的值："+user.getAge()+"===再取一下hashCode:"user.hashCode());
        user.setAge(1);
        System.out.println("普通变量var:" + (++var) + "===静态变量staticVar:" + (++staticVar) +===配置变量testInt:" + (++testInt)
                + "===ThreadLocal变量tl:" + tl.get()+"===注入变量user:" + user.getAge());
```

```
        return "普通变量var:" + var + ",静态变量staticVar:" + staticVar + ",配置读取变量testInt:" + t
stInt + ",ThreadLocal变量tl:"
                + tl.get() + "注入变量user:" + user.getAge();
    }
}
```

补充Controller以外的代码，config里面自己定义的Bean:User

```
@Configuration
public class MyConfig {
    @Bean
    public User user(){
        return new User();
    }
}
```

我暂时能想到的定义变量的方法就这么多了，三次http请求结果如下：

```
先取一下user对象中的值：0===再取一下hashCode:241165852
普通变量var:1===静态变量staticVar:1===配置变量testInt:1===ThreadLocal变量tl:1===注入变
user:1
先取一下user对象中的值：1===再取一下hashCode:241165852
普通变量var:2===静态变量staticVar:2===配置变量testInt:2===ThreadLocal变量tl:1===注入变
user:1
先取一下user对象中的值：1===再取一下hashCode:241165852
普通变量var:3===静态变量staticVar:3===配置变量testInt:3===ThreadLocal变量tl:1===注入变
user:1
```

可以看到，在单例模式下Controller中只有用ThreadLocal封装的变量是线程安全的。为什么这样说？

我们可以看到3次请求结果里面只有ThreadLocal变量值每次都是从0+1=1的，其他的几个都是累加，而user对象呢，默认值是0，第二交取值的时候就已经是1了，关键他的hashCode是一样的，说明次请求调用的都是同一个user对象。

下面将TestController 上的@Scope注解的属性改一下改成多实例的：@Scope(value = "prototype"，其他都不变，再次请求，结果如下：

```
先取一下user对象中的值：0===再取一下hashCode:853315860
普通变量var:1===静态变量staticVar:1===配置变量testInt:1===ThreadLocal变量tl:1===注入变
user:1
先取一下user对象中的值：1===再取一下hashCode:853315860
普通变量var:1===静态变量staticVar:2===配置变量testInt:1===ThreadLocal变量tl:1===注入变
user:1
先取一下user对象中的值：1===再取一下hashCode:853315860
普通变量var:1===静态变量staticVar:3===配置变量testInt:1===ThreadLocal变量tl:1===注入变
user:1
```

分析这个结果发现，多实例模式下普通变量，取配置的变量还有ThreadLocal变量都是线程安全的，静态变量和user（看他的hashCode都是一样的）对象中的变量都是非线程安全的。

也就是说尽管TestController 是每次请求的时候都初始化了一个对象，但是静态变量始终是只有一份，而且这个注入的user对象也是只有一份的。静态变量只有一份这是当然的咯，那么有没有办法让use对象可以每次都new一个新的呢？当然可以：

```
public class MyConfig {
    @Bean
    @Scope(value = "prototype")
    public User user(){
        return new User();
    }
}
```

在config里面给这个注入的Bean加上一个相同的注解@Scope(value = "prototype")就可以了，再请求一下看看：

```
先取一下user对象中的值：0===再取一下hashCode:1612967699
普通变量var:1===静态变量staticVar:1===配置变量testInt:1===ThreadLocal变量tl:1===注入变
user:1
先取一下user对象中的值：0===再取一下hashCode:985418837
普通变量var:1===静态变量staticVar:2===配置变量testInt:1===ThreadLocal变量tl:1===注入变
user:1
先取一下user对象中的值：0===再取一下hashCode:1958952789
普通变量var:1===静态变量staticVar:3===配置变量testInt:1===ThreadLocal变量tl:1===注入变
user:1
```

可以看到每次请求的user对象的hashCode都不是一样的，每次赋值前取user中的变量值也都是默认0。

小结

1. 在@Controller/@Service等容器中，默认情况下，scope值是单例-singleton的，也是线程不安的。

2. 尽量不要在@Controller/@Service等容器中定义静态变量，不论是单例(singleton)还是多实例(prototype)他都是线程不安全的。

3. 默认注入的Bean对象，在不设置scope的时候他也是线程不安全的。

4. 一定要定义变量的话，用ThreadLocal来封装，这个是线程安全的

来源：https://www.cnblogs.com/myseries/p/11729800.html

作者：myseries