Spring Boot中的多线程问题和ThreadLocal 原

1. Spring中的多线程疑惑

首先我们需要认清：

1. web容器本身就是**多线程**的，**每一个HTTP请求都会产生一个独立的线程**（或者从线程池中取得创建好的线程）；
2. Spring中的bean（用@Repository、@Service、@Component和@Controller注册的bean）都是**单例**的，即整个程序、所有线程**共享一个实例**；
3. 虽然bean都是单例的，但是**Spring提供的模板类（XXXTemplate）**，在Spring容器的管理下（使用@Autowired注入），**会自动使用ThreadLocal以实现多线程；**
4. **即类是单例的，但是其中有可能出现并发问题的变量使用ThreadLocal实现了多线程。**
5. 注意除了Spring本身提供的类以外，在Bean中定义“有状态的变量”（即有存储数据的变量），**其会被所有线程共享**，很可能导致并发问题，**需要自行另外使用ThreadLocal进行处理，或者将Bean声明为prototype型。**
6. 一个类中的方法实际上是独立，方法内定义的局部变量在每次调用方法时都是独立的，不会有并发问题。只有类的“有状态的”全局变量会有并发问题

结论：

1. **使用Spring提供的template等类没有多线程问题！**
2. **一般来说只有类的属性/全局变量会导致多线程问题，而方法内的局部变量不会有并发问题**
3. **单例模式肯定是线程不安全的！ spring的Bean中的自定义的成员变量除非进行threadlocal封装，否则都是非线程安全的！**

2. Spring中的prototype和@Autowired

2.1 使用@Autowired没有实现多个实例

注意即使使用@Scope(BeanDefinition.SCOPE_PROTOTYPE)将Bean声明为prototype，如果：

1. 外层的类是singleton
2. 使用@Autowired注入

这样的话**仍然只会有一个实例。**例如：

使用@Scope(BeanDefinition.SCOPE_PROTOTYPE)声明的Bean

```
@Component
@Scope(BeanDefinition.SCOPE_PROTOTYPE)
public class ClassB {

    private Integer num = Integer.valueOf(0);

    public Integer getNum() {
        return num;
    }

```

```
    public void setNum(Integer num) {
        this.num = num;
    }
}
```

使用@Autowired注入到ClassA中

```
/**
 * Created by ASUS on 2017/5/24.
 */
@Component
public class ClassA {
    @Autowired
    private ClassB classB;

    public Integer addNum(){
        classB.setNum(classB.getNum()+1);
        System.out.println(classB.getNum());
        return classB.getNum();
    }
}
```

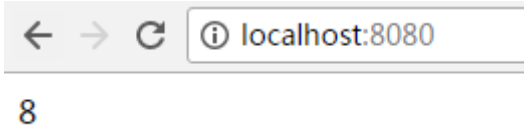
通过Controller调用，用@Autowire将ClassA注入。

```
@RestController
public class Controller {

    @Autowired
    private ClassA classA;

    @RequestMapping("/")
    public String print(){
        return classA.addNum().toString();
    }
}
```

每一次访问都会导致num+1，访问8次后：



并没有实现"多个实例"的效果，一次都是在操作同一个ClassB实例。这是因为使用@Autowired实际上和直接new是一个效果，只是交由Spring容器实现而已。而ClassA本身是一个单例，单例只会实例化一次，这样其属性自然也就只会被实例化一次。

2.2 解决方法

2.2.1 直接在方法中声明局部变量

在Java或者其他语言中，每个“方法”在被调用时，都会重新声明一遍方法中的局部变量。如果想要classB为多实例，直接在方法中声明即可。比如：

```
@Component
@Scope(BeanDefinition.SCOPE_PROTOTYPE)
public class ClassA {
    public Integer addNum(){
        //不使用@Autowired，直接在方法中声明
        ClassB classB = new ClassB();
        classB.setNum(classB.getNum()+1);
        System.out.println(classB.getNum());
        return classB.getNum();
    }
}
```

注意：

这样做的问题在于，如果ClassB中需要使用@Autowired，则这个@Autowired会失效。

比如想在ClassB中使用Spring管理的JdbcTemplate，就需要使用@Autowired。如果ClassB不是通过@Autowired实例化的，ClassB中的JdbcTemplate就会注入失败，导致NullPointerException。

2.2.2 使用ThreadLocal管理属性

还要一个方法实现和多实例“类似”的功能，即使用ThreadLocal来管理类中的属性。例如对于上面的例子：

```
@Component
@Scope(BeanDefinition.SCOPE_PROTOTYPE)
public class ClassB {

    //使用ThreadLocal管理属性，每个线程都操作一个新的副本
    private static ThreadLocal<Integer> integerThreadLocal = new
ThreadLocal<Integer>(){
        @Override
        protected Integer initialValue() {
            return 0;
        }
    };

    public ThreadLocal<Integer> getIntegerThreadLocal() {
        return integerThreadLocal;
    }
}
```

在ClassB中就可以正常使用@Autowired进行注入。

```java
@Component
@Scope(BeanDefinition.SCOPE_PROTOTYPE)
public class ClassA {

    @Autowired
    ClassB classB;

    public Integer addNum(){
        Integer integer = classB.getIntegerThreadLocal().get();
        integer++;
        return integer;
    }
}
```