springboot启动设置初始化数据

题目标签

- 题目难度：一般
- 知识点标签：springboot
- 课程时长：20分钟

题目描述

面试中的问题：springboot如何在启动时设置初始化数据

案情回顾

在实际开发中，我们一般都会在配置文件（application.properties或者application.yml）中配置各个项目中集成的属性值，来进行各个组件的设置，比如配置端口：server.port=8080，但是在开发中我们需要配置业务得属性值来添加到springboot容器中，那么这个时候我们应该怎么办呢?

解决方案

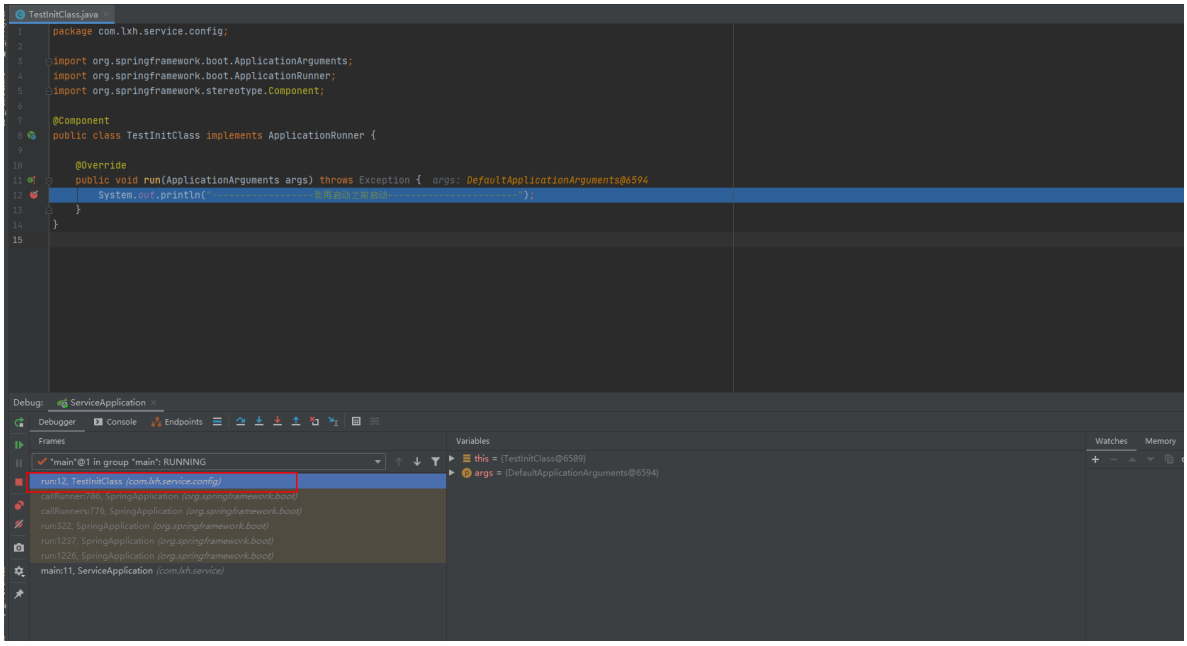
实现`ApplicationRunner`接口

编写测试类

```
@Component
public class TestInitClass implements ApplicationRunner {

    @Override
    public void run(ApplicationArguments args) throws Exception {
        System.out.println("------------------我在启动之前加载了------------------
----");
    }
}
```

演示测试

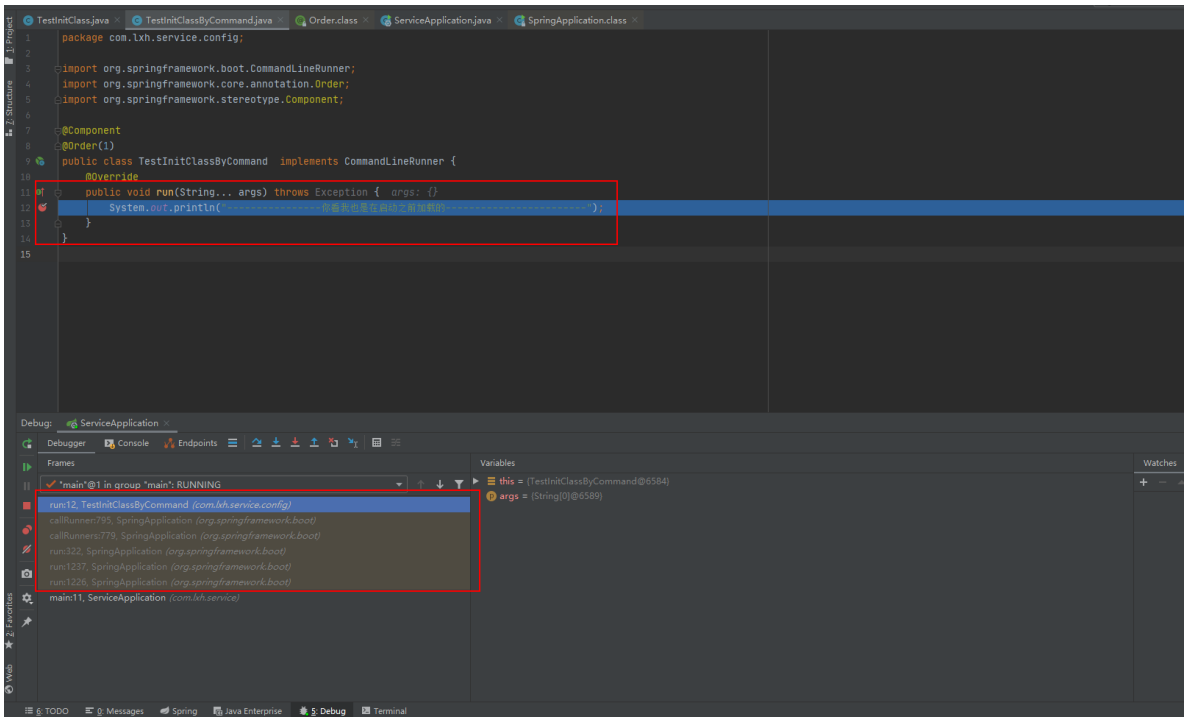


实现CommandLineRunner 接口

编写测试类

```
@Component
public class TestInitClassByCommand  implements CommandLineRunner {
    @Override
    public void run(String... args) throws Exception {
        System.out.println("-----------------你看我也是在启动之前加载的----------------
----------");
    }
}
```

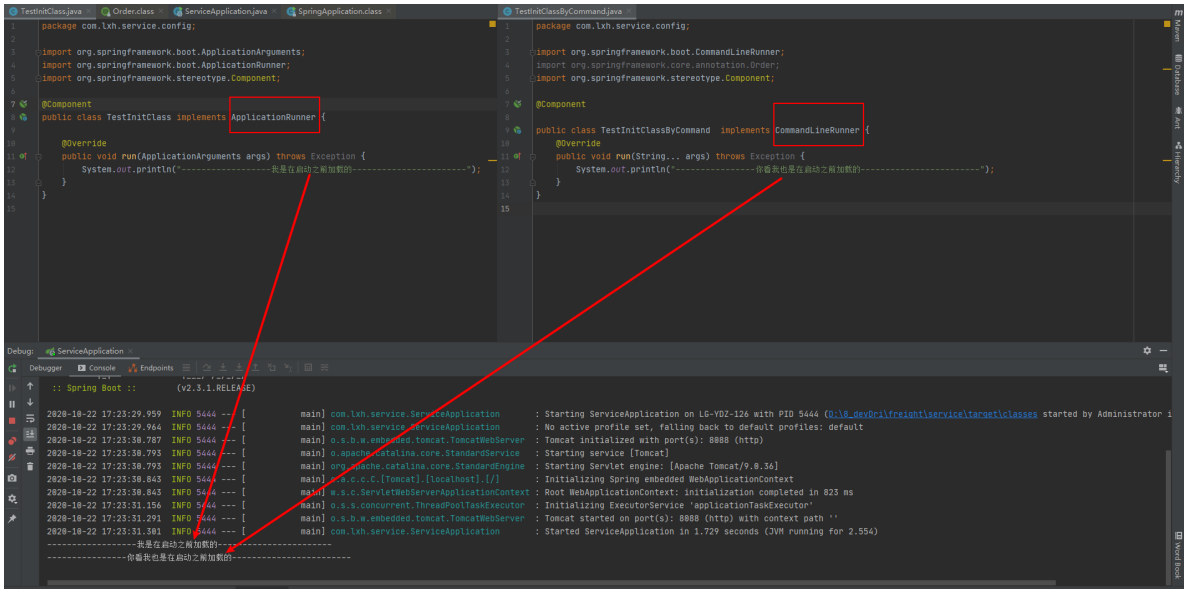

演示测试



优先级

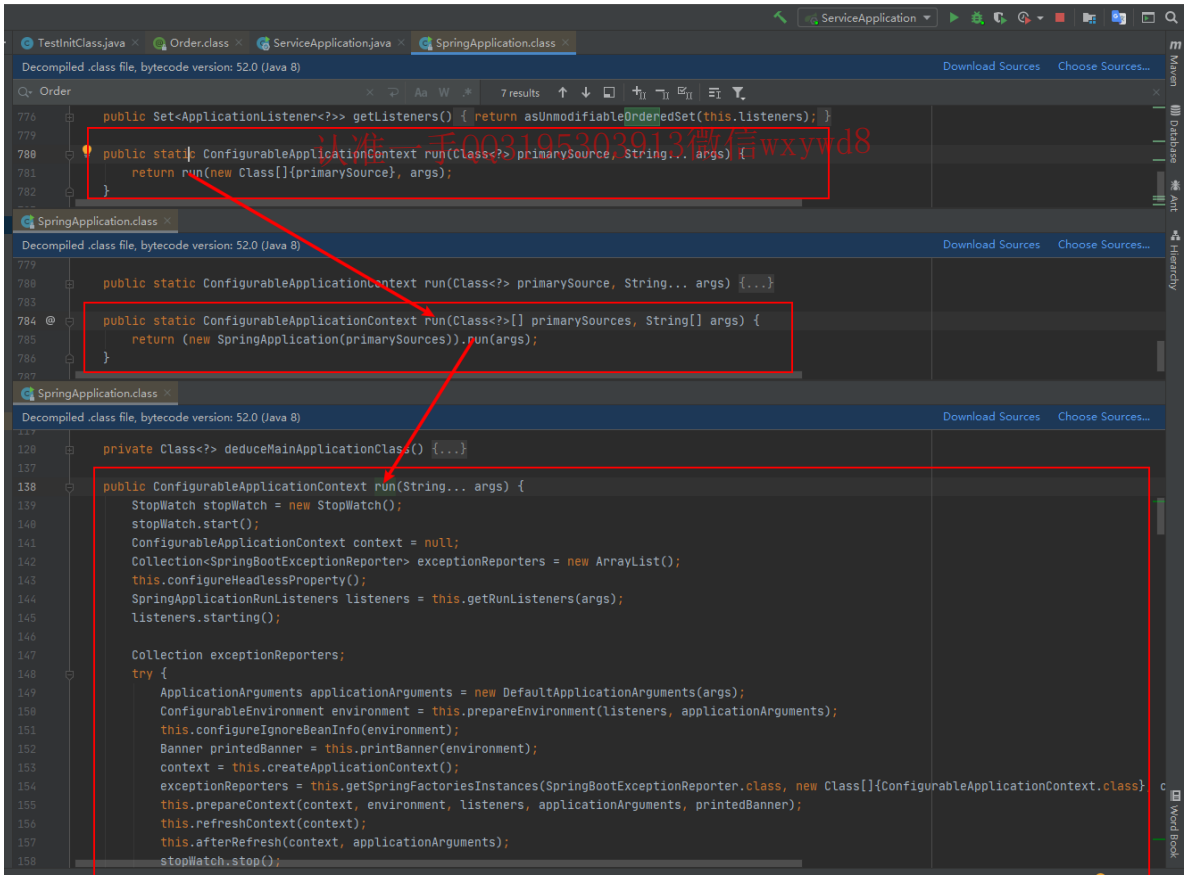
默认情况下ApplicationRunner比CommandLineRunner 先执行

首先我们运行验证一下：

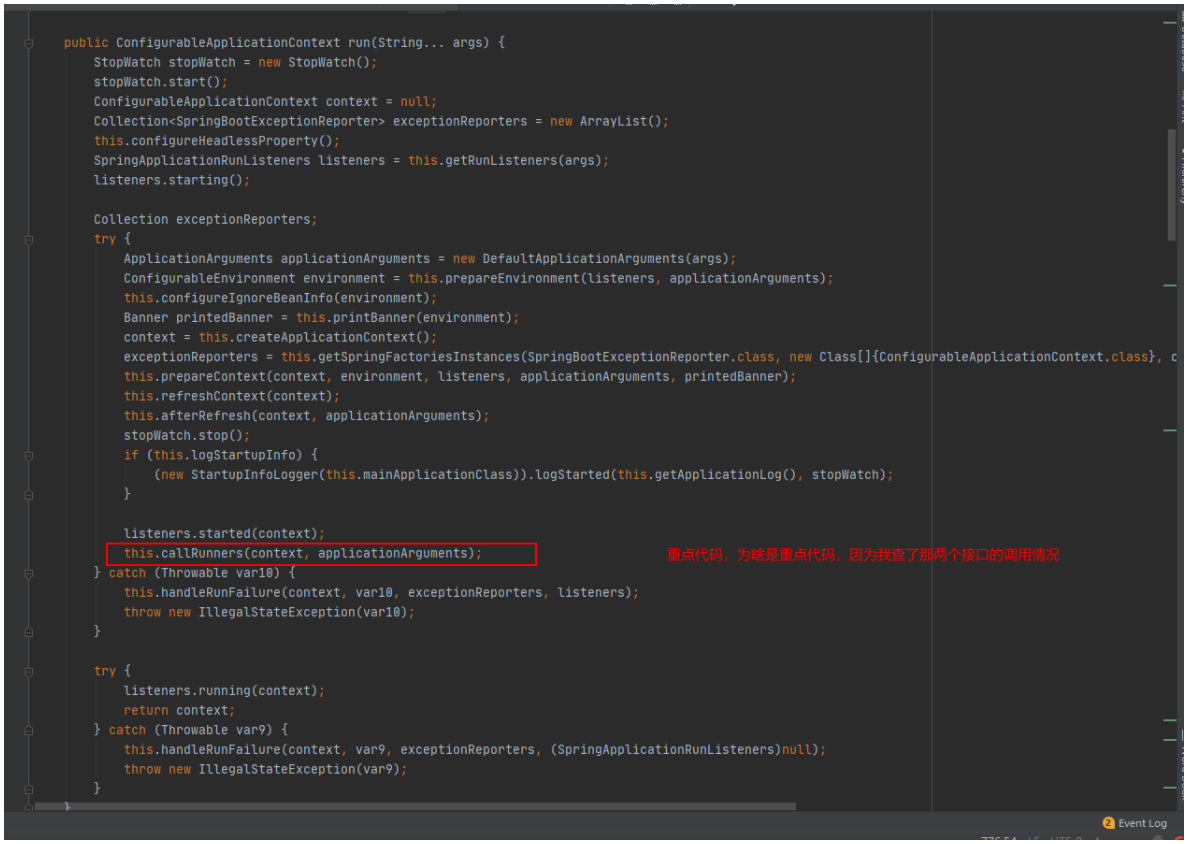


验证成功。

进行源码查看为什么会这样呢，我们在启动类中点击run()方法进入该方法的内容里面，该方法为springApplication类的方法，调用过程如下



由此我们可以看到run()方法的执行，最终是走的138行的run方法。那么我们具体看下最后这个run方法



我们继续往下跟，查看该callRunners方法里具体的执行情况

```
    private void callRunners(ApplicationContext context, ApplicationArguments
args) {
        List<Object> runners = new ArrayList();
        //重点来了，看这个添加，这个添加是先添加 ApplicationRunner接口的类型再添加
CommandLineRunner 接口的类型。

 runners.addAll(context.getBeansOfType(ApplicationRunner.class).values());

 runners.addAll(context.getBeansOfType(CommandLineRunner.class).values());
        AnnotationAwareOrderComparator.sort(runners);
        Iterator var4 = (new LinkedHashSet(runners)).iterator();

        //循环时也是先进行了ApplicationRunner类型的判断和调用。
        while(var4.hasNext()) {
            Object runner = var4.next();
            if (runner instanceof ApplicationRunner) {
                this.callRunner((ApplicationRunner)runner, args);
            }

            if (runner instanceof CommandLineRunner) {
                this.callRunner((CommandLineRunner)runner, args);
            }
        }

    }
```

所以，我们应该能明白，调用的顺序原因了。那么我们是否能够手动指定谁先加载，谁在加载呢，答案是肯定的。同样这个callRunners方法，里面有一句是这么写的

`AnnotationAwareOrderComparator.sort(runners);`这个是说明我会先进行一下排序。那么怎么排序的呢

```
//AnnotationAwareOrderComparator.java sort方法
public static void sort(List<?> list) {
        if (list.size() > 1) {
            list.sort(INSTANCE);
        }
    }
```

经过简单查看，发现他是走的list的排序。然后我们回来看注解类 `Order`

```
/**
 *@Order定义了已注释组件的排序顺序。
 *该value是可选的，并且表示为在定义的顺序值Ordered接口。 值越低，具有更高的优先级。 默认值是
Ordered.LOWEST_PRECEDENCE ，
 *表示最低优先级（输给任何其他指定顺序值）
 * @author Rod Johnson
 * @author Juergen Hoeller
 * @since 2.0
 * @see org.springframework.core.Ordered
 * @see AnnotationAwareOrderComparator
 * @see OrderUtils
 * @see javax.annotation.Priority
 */
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE, ElementType.METHOD, ElementType.FIELD})
@Documented
public @interface Order {

    /**
     * 排序值
     默认为Ordered类中的LOWEST_PRECEDENCE
     * <p>Default is {@link Ordered#LOWEST_PRECEDENCE}.
     * @see Ordered#getOrder()
     */
    int value() default Ordered.LOWEST_PRECEDENCE;

}
```


综上：我们可以知道当使用注解Order时给的参数值越小则有更好的优先级。

总结

- 所有CommandLineRunner/ApplicationRunner的执行时点是在SpringBoot应用的ApplicationContext完全初始化开始工作之后，`callRunners()`可以看出是run方法内部最后一个调用的方法(可以认为是main方法执行完成之前最后一步)
- 只要存在于当前SpringBoot应用的ApplicationContext中的任何CommandLineRunner/ApplicationRunner，都会被加载执行(不管你是手动注册还是自动扫描去Ioc容器)
- 使用@Order注解可以设置加载的顺序
- 一般情况我们只需要使用一个接口来加载初始化数据即可