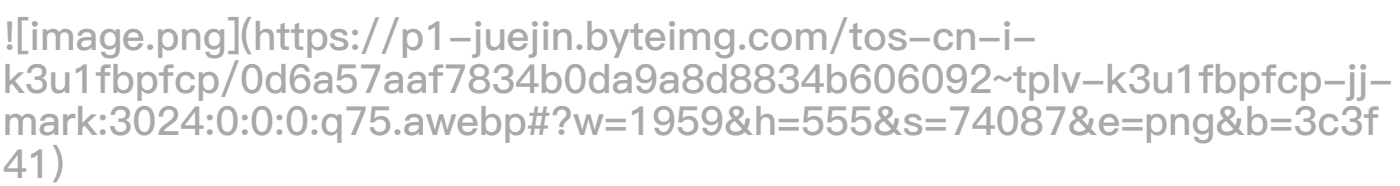
实战指南：四种调整 Spring Bean 初始化顺序的方案

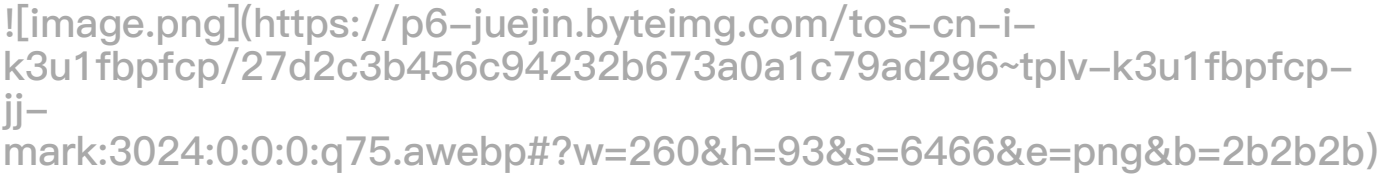
背景

因为业务需求，mentor想要某些 bean 启动时优先加载，将数据存入缓存，便问我，“能不能调下Bean初始化顺序？”，于是便有了这篇文章

结构演示



目前一共有两个 **service** ，每个 **service** 都有一个 **init** 方法，打印bean创建时机，正常状态打印结果如下：

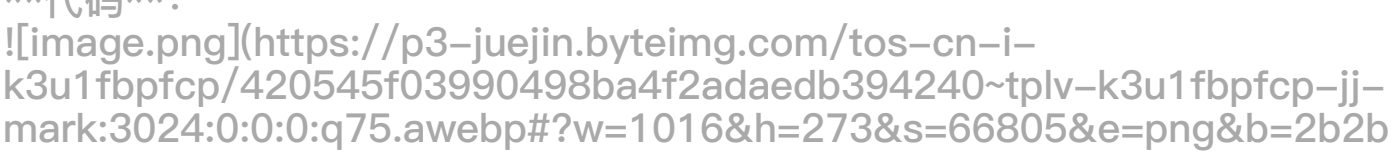


正文

方案一（ @Order ）

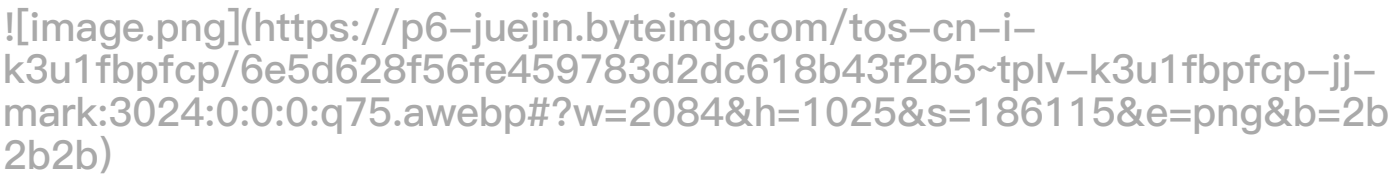
这是第一个想到的方法，我们给每个service上加上@Order，让他们倒序创建

代码：



2b)

结果:



嗯？不是数字越低优先级越高吗，结果怎么还是 **1 -> 2**？相信眼尖的人已经看出来了，我在开头埋了个坑，用 **@PostConstruct** 作初始化操作

题外话(@PostConstruct 和 @Order 优先级)

* **@PostConstruct** 修饰的方法会在服务器加载Servlet的时候运行，并且只会被服务器执行一次，也就在依赖注入完成后立即调用，bean 初始化阶段执行的。
* **@Order** 注解用于设置组件的执行顺序，排序集合或指定某些类型的组件的优先级

@PostConstruct 方法的执行顺序是由 **Spring** 容器在 bean 初始化过程中自动管理的，与 **@Order** 注解无关。

从 **SpringBoot** 的 **run** 源码角度来看

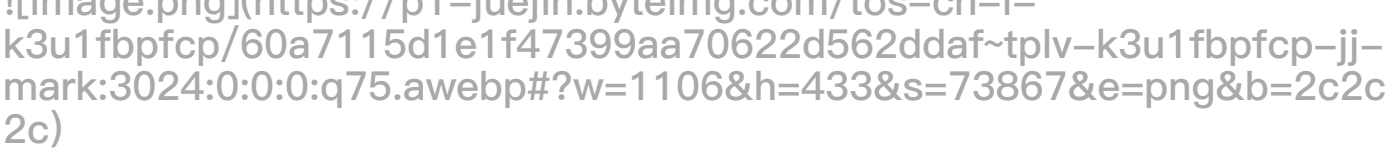

结论

优点 ： 简单明了，适用于需要简单控制初始化顺序的场景。

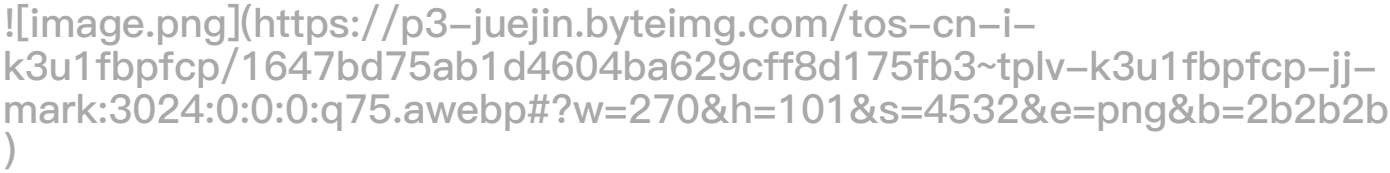
缺点：只适用于具有顺序的Bean，无法处理复杂依赖关系，遇到如 @PostConstruct 会失效

方案二（ SmartInitializingSingleton ）

SmartInitializingSingleton 用于在容器完成所有**单例 bean** 的初始化后执行一些额外的初始化工作。用这个接口应该能保证 FirstService 后创建了吧

代码:


结果:



还是**不行**，**@PostConstruct** 的东西还是先创建了，不过起码保证了 **FirstService** 的 **afterSingletonsInstantiated** 方法是所有单例 Bean 初始化之后执行的。

不过毫无疑问，被驳回了，还说这要是有三个或多个 **Bean** 有这业务怎么办

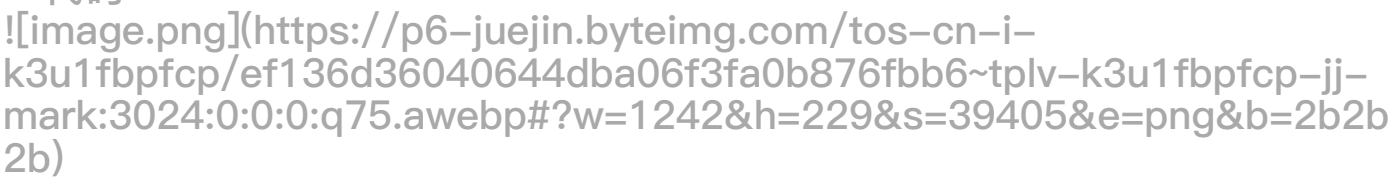
结论

优点 ： 确保所有单例bean都初始化，适合在所有Bean创建后执行全局初始化逻辑。

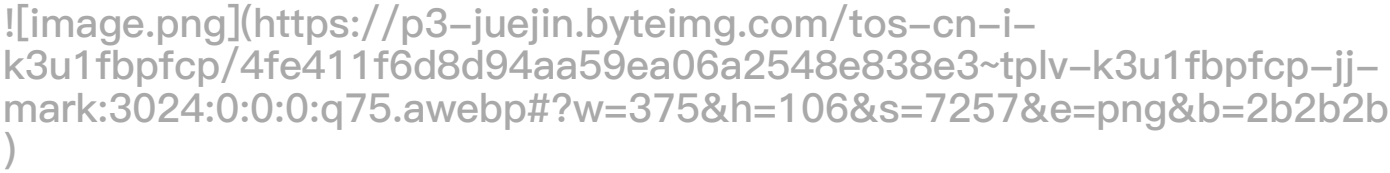
缺点：不适合控制特定Bean之间的初始化顺序。

方案三（ @DependsOn ）

既然加载顺序不行，还要有多个 Bean ，那就从 Bean 间的依赖入手嘛

代码:


结果:



不出意外的成功了，但是mentor嫌耦合性太高，一处改了处处改，后期项目大了找不到不方便维护

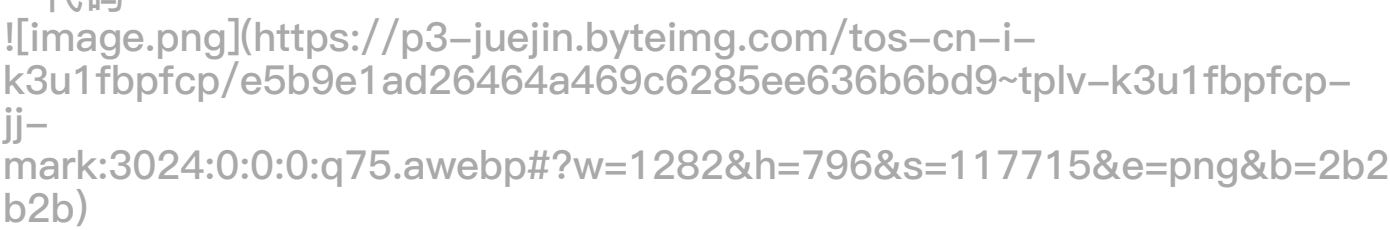
结论

优点 ： 简单明了，适用于明确的依赖关系。

缺点：依赖关系硬编码在配置类中，灵活性较低，耦合性高。

方案四（ 自定义 Bean 初始化类 ）

那好嘛，那自定义呗

代码


在 META-INF 的 spring.factories 加上配置

```
org.springframework.context.ApplicationContextInitializer=\com.hhh.init.MyBeanInit
```

解释：

1. 该自定义类 **MyBeanInit** 分别继承了 **BeanDefinitionRegistryPostProcessor** 和 **ApplicationContextInitializer** ，重写其内部方法
* **BeanDefinitionRegistryPostProcessor** ： **提前注册或修改 Bean 定义**，在所有其他 `BeanFactoryPostProcessor` 运行之前执行，允许我们注册或修改 `BeanDefinition`
* **ApplicationContextInitializer** ： **初始化应用上下文**，在应用上下文刷新之前调用，可以动态地为应用上下文添加属性、`BeanFactoryPostProcessor` 或其他配置。
2. 在 **postProcessBeanDefinitionRegistry** 方法中，使用 BeanDefinitionBuilder 创建了两个 AbstractBeanDefinition 实例，分别对应 `ThirdService` 和 `SecondService` 类，然后，将这些 Bean 定义注册到 Spring 容器中，分别命名为 `"thirdService"` 和 `"secondService"`。
3. 在 **initialize** 方法中，将当前的 BeanDefinitionRegistryPostProcessor 实例（ **MyBeanInit** 自身）添加到应用上下文的 `BeanFactoryPostProcessor` 列表中。在容器启动时，`postProcessBeanDefinitionRegistry` 方法将被调用，从而注册我们在上面定义的 Bean。

简而言之：自定义 Bean **注册方法**，将自己想要优先加载的 Bean 塞进去，再加入**上下文**中加载

这里我们也可以把 **@Component** 去掉，因为在自定义初始化类中加载了，不需要被 **ComponentScan** 再扫描注册一次，以免出现重复注册异常

(https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/2a485853eebe4f90b4d39bcd31ee0b0d~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=955&h=256&s=44010&e=png&b=2c2c2c)

结果

(https://p9-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/ac6eef868bd045c78eb26edae67edad4~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=415&h=103&s=6252&e=png&b=2b2b2b)

结论

优点：灵活性高，可以用于复杂的初始化逻辑。

缺点：需要手动管理Bean的初始化顺序，代码维护成本较高。

结尾
==

他说不用了

原文链接: https://juejin.cn/post/7380663251632586767