https://blog.csdn.net/neweastsun/article/details/78775371

**使用Spring @DependsOn控制bean加载顺序**

2017年12月11日 18:26:39 [neweastsun](https://me.csdn.net/neweastsun" \t "_blank) 阅读数 30790

版权声明：本文为博主原创文章，遵循[CC 4.0 by-sa](http://creativecommons.org/licenses/by-sa/4.0/" \t "_blank)版权协议，转载请附上原文出处链接和本声明。

本文链接：<https://blog.csdn.net/neweastsun/article/details/78775371>

**使用Spring @DependsOn控制bean加载顺序**

spring容器载入bean顺序是不确定的，spring框架没有约定特定顺序逻辑规范。但spring保证如果A依赖B(如beanA中有@Autowired B的变量)，那么B将先于A被加载。但如果beanA不直接依赖B，我们如何让B仍先加载呢？

**控制bean初始化顺序**

可能有些场景中，bean A 间接依赖 bean B。如Bean B应该需要更新一些全局缓存，可能通过单例模式实现且没有在spring容器注册，bean A需要使用该缓存；因此，如果bean B没有准备好，bean A无法访问。

另一个场景中，bean A是事件发布者（或JMS发布者），bean B (或一些) 负责监听这些事件，典型的如观察者模式。我们不想B 错过任何事件，那么B需要首先被初始化。

简言之，有很多场景需要bean B应该被先于bean A被初始化，从而避免各种负面影响。我们可以在bean A上使用@DependsOn注解，告诉容器bean B应该先被初始化。下面通过示例来说明。

**示例说明**

示例通过事件机制说明，发布者和监听者，然后通过spring配置运行。为了方便说明，示例进行了简化。

**EventManager.java**   
事件管理类，维护监听器列表，通过单例方法获取事件管理器，可以增加监听器或发布事件。

import java.util.ArrayList;

import java.util.List;

import java.util.function.Consumer;

public class EventManager {

private final List<Consumer<String>> listeners = new ArrayList<>();

private EventManager() {

}

private static class SingletonHolder {

private static final EventManager INSTANCE = new EventManager();

}

public static EventManager getInstance() {

return SingletonHolder.INSTANCE;

}

public void publish(final String message) {

listeners.forEach(l -> l.accept(message));

}

public void addListener(Consumer<String> eventConsumer) {

listeners.add(eventConsumer);

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27

**EventPublisherBean.java**   
事件发布类，通过EventManager类发布事件。

import com.logicbig.example.EventManager;

public class EventPublisherBean {

public void initialize() {

System.out.println("EventPublisherBean initializing");

EventManager.getInstance().publish("event published from EventPublisherBean");

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

**EventListenerBean.java**   
事件监听者，可以增加监听器。

import com.logicbig.example.EventManager;

public class EventListenerBean {

private void initialize() {

EventManager.getInstance().

addListener(s ->

System.out.println("event received in EventListenerBean : " + s));

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10

**AppConfig.java**   
配置运行类。

@Configuration

@ComponentScan("com.logicbig.example")

public class AppConfig {

@Bean(initMethod = "initialize")

@DependsOn("eventListener")

public EventPublisherBean eventPublisherBean () {

return new EventPublisherBean();

}

@Bean(name = "eventListener", initMethod = "initialize")

// @Lazy

public EventListenerBean eventListenerBean () {

return new EventListenerBean();

}

public static void main (String... strings) {

new AnnotationConfigApplicationContext(AppConfig.class);

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20

运行AppConfig的main方法，输出结果为：

EventListenerBean initializing

EventPublisherBean initializing

event received in EventListenerBean : event published from EventPublisherBean

* 1
* 2
* 3

**总结**

如果我们注释掉@DependsOn("eventListener")，我们可能不确定获得相同结果。尝试多次运行main方法，偶尔我们将看到EventListenerBean 没有收到事件。为什么是偶尔呢？因为容器启动过程中，spring按任意顺序加载bean。

那么当不使用@DependsOn可以让其100%确定吗？可以使用@Lazy注解放在eventListenerBean ()上。因为EventListenerBean在启动阶段不加载，当其他bean需要其时才加载。这次我们仅EventListenerBean被初始化。

EventPublisherBean initializing

* 1

现在从新增加@DependsOn，也不删除@Lazy注解，输出结果和第一次一致，虽然我们使用了@Lazy注解，eventListenerBean在启动时仍然被加载，因为@DependsOn表明需要EventListenerBean。