

Spring注解驱动开发第38讲——你知道ApplicationListener的用法吗?

写在前面

在前面两讲中，我们学习了一下Spring扩展原理里面的BeanFactoryPostProcessor和BeanDefinitionRegistryPostProcessor。在这一讲中，我们将会学习一下Spring剩余其他扩展原理里面的ApplicationListener。

你不知道的ApplicationListener

ApplicationListener的概述

ApplicationListener按照字面意思，它应该是Spring里面的应用 **监听器**，也就是Spring为我们提供的基于事件驱动开发的功能。

接下来，我们看一下ApplicationListener的源码，如下图所示，可以看到它是一个接口。



```
ApplicationListener.class
2+ * Copyright 2002-2015 the original author or authors.
16
17 package org.springframework.context;
18
19 import java.util.EventListener;
20
21 /**
22  * Interface to be implemented by application event listeners.
23  * Based on the standard {@code java.util.EventListener} interface
24  * for the Observer design pattern.
25  *
26  * <p>As of Spring 3.0, an ApplicationListener can generically declare the event type
27  * that it is interested in. When registered with a Spring ApplicationContext, events
28  * will be filtered accordingly, with the listener getting invoked for matching event
29  * objects only.
30  *
31  * @author Rod Johnson
32  * @author Juergen Hoeller
33  * @param <E> the specific ApplicationEvent subclass to listen to
34  * @see org.springframework.context.event.ApplicationEventMulticaster
35  */
36 public interface ApplicationListener<E extends ApplicationEvent> extends EventListener {
37
38     /**
39      * Handle an application event.
40      * @param event the event to respond to
41      */
42     void onApplicationEvent(E event);
43
44 }
45
```

也就是说，如果我们要写一个监听器，那么我们要写的监听器就得实现这个接口，而该接口中带的泛型就是我们要监听的事件。也就是说，我们应该要监听ApplicationEvent及其下面的子事件，因此，如果我们要发布事件，那么所发布的事件应该是ApplicationEvent的子类。

ApplicationListener的作用

它的作用主要是来监听IOC容器中发布的一些事件，只要事件发生便会来触发该监听器的回调，从而来完成事件驱动模型的开发。

ApplicationListener的用法

首先，编写一个类来实现ApplicationListener接口，例如MyApplicationListener，这实际上就是写了一个监听器。

```
1 package com.meimeixia.ext;
2
3 import org.springframework.context.ApplicationEvent;
4 import org.springframework.context.ApplicationListener;
5 import org.springframework.stereotype.Component;
6
7 // 当然了，监听器这东西要工作，我们还得把它添加在容器中
8 @Component
9 public class MyApplicationListener implements ApplicationListener<ApplicationEvent> {
10
11     // 当容器中发布此事件以后，下面这个方法就会被触发
12     @Override
13     public void onApplicationEvent(ApplicationEvent event) {
14
15     }
```

```
15 // TODO Auto-generated method stub
16 System.out.println("收到事件: " + event);
17 }
18 }
```

AI写代码java运行



然后，我们就要来测试一下以上监听器的功能了。试着运行IOCTest_Ext测试类中的test01方法，看能不能收到事件？

```
1 package com.meimeixia.test;
2
3 import org.junit.Test;
4 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
5
6 import com.meimeixia.ext.ExtConfig;
7
8 public class IOCTest_Ext {
9
10     @Test
11     public void test01() {
12         AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(ExtConfig.class);
13
14         // 关闭容器
15         applicationContext.close();
16     }
17 }
18 }
```

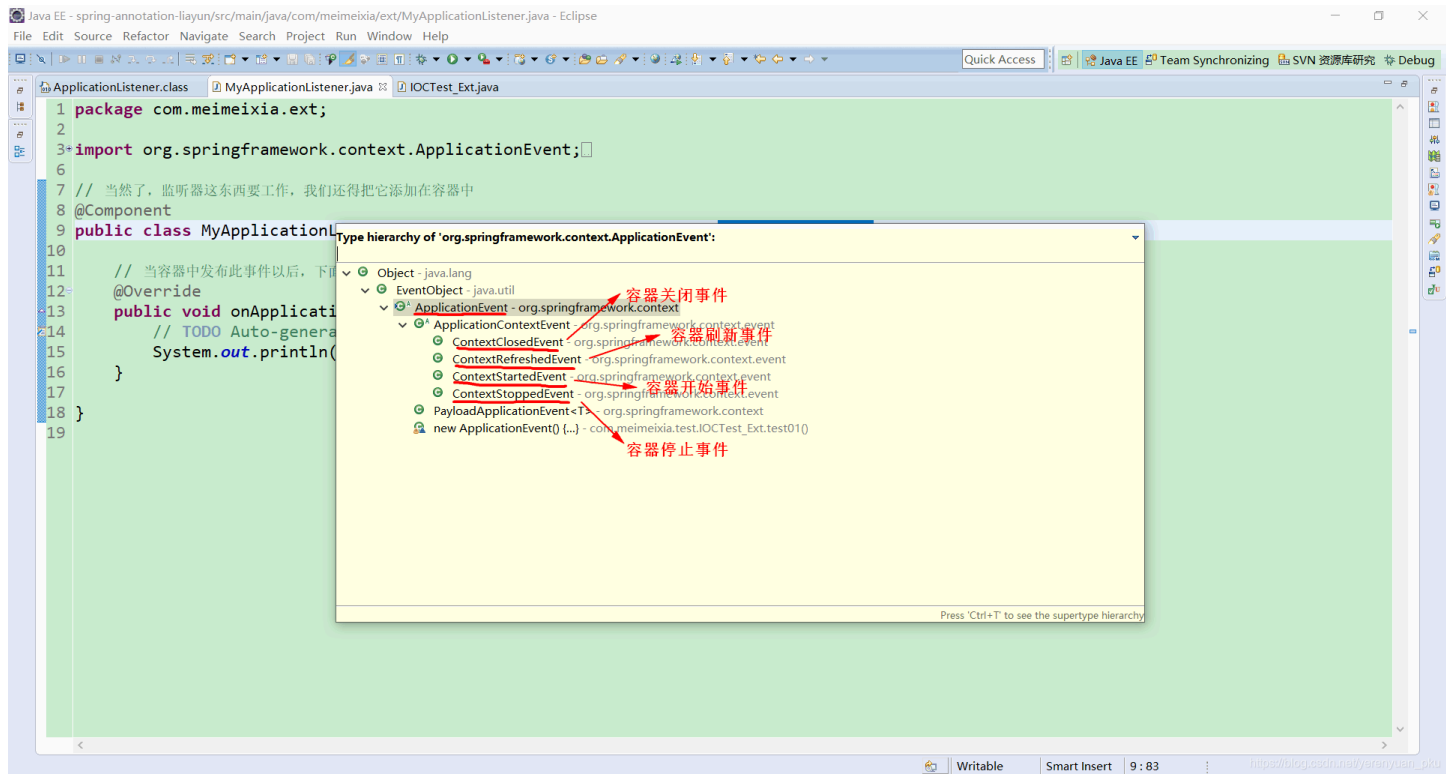
AI写代码java运行



如果运行以上test01方法，那么你会看到Eclipse 控制台打印出了如下内容。

```
<terminated> IOCTest_Ext.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2021年2月12日 下午9:32:25)
二月 12, 2021 9:32:27 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepareRefresh
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@21588809: startup date [Fri Feb 12 21:32:27 CST 2021]
postProcessBeanDefinitionRegistry...bean的数量: 11
MyBeanDefinitionRegistryPostProcessor...bean的数量: 12
MyBeanFactoryPostProcessor...postProcessBeanFactory...
当前BeanFactory中有12个Bean
[org.springframework.context.annotation.internalConfigurationAnnotationProcessor, org.springframework.context.annotation.internalAutowiredAnnotationProcessor]
二月 12, 2021 9:32:28 下午 org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor <init>
信息: JSR-330 'javax.inject.Inject' annotation found and supported for autowiring
blue...constructor
blue...constructor
收到事件: 1 org.springframework.context.event.ContextRefreshedEvent[source=org.springframework.context.annotation.AnnotationConfigApplicationContext@21588809]
二月 12, 2021 9:32:28 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext doClose
信息: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@21588809: startup date [Fri Feb 12 21:32:27 CST 2021]; root of context hierarchy
收到事件: 2 org.springframework.context.event.ContextClosedEvent[source=org.springframework.context.annotation.AnnotationConfigApplicationContext@21588809]
```

哎，可以看到我们收到了两个事件，这两个事件分别是 `org.springframework.context.event.ContextRefreshedEvent` 和 `org.springframework.context.event.ContextClosedEvent`，其中第一个是容器已经刷新完成事件，第二个是容器关闭事件。而且，从下图中可以看到，这两个事件都是 `ApplicationEvent` 下面的事件。



只不过现在暂时还没用到容器开始和容器停止这两个事件而已。其实，想必你也已经猜到了，IOC容器在刷新完成之后便会发布ContextRefreshEvent事件，一旦容器关闭了便会发布ContextClosedEvent事件。

这时，你不禁要问了，我们可不可以自己发布事件呢？当然可以了，只不过此时我们应该遵循如下的步骤来进行开发。

第一步，写一个监听器来监听某个事件。当然了，监听的这个事件必须是ApplicationEvent及其子类。

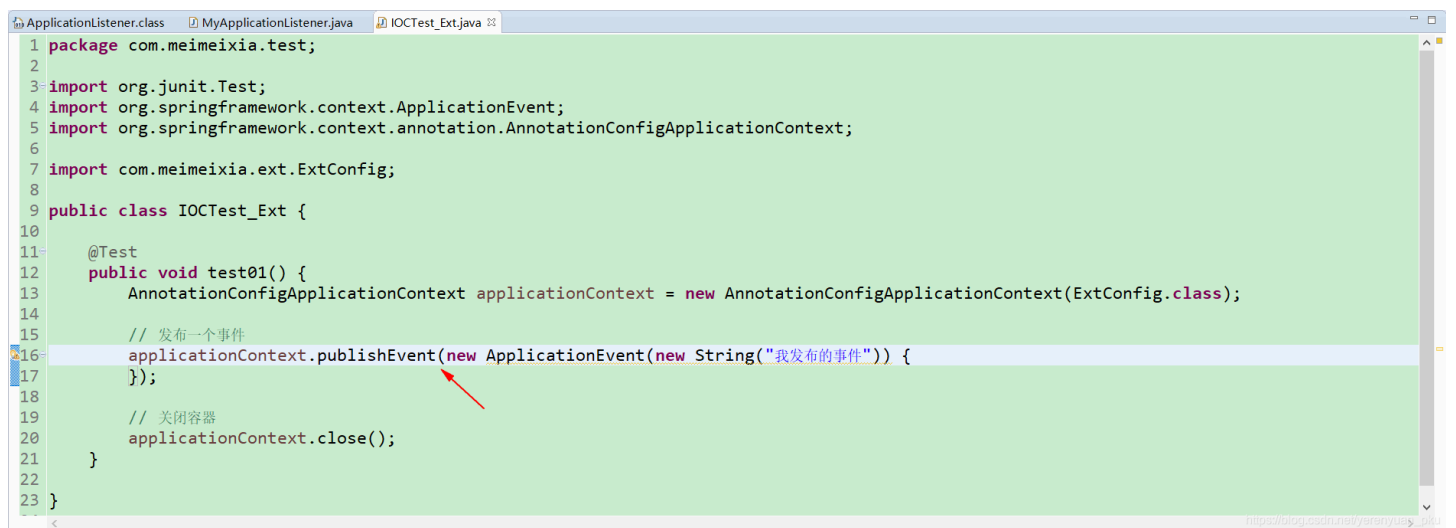
第二步，把监听器加入到容器中，这样Spring才能知道有这样一个监听器。

第三步，只要容器中有相关事件发布，那么我们就能监听到这个事件。举个例子，就拿我们上面监听的两个事件来说，你要搞清楚的一个问题是谁发布了这两个事件，猜都能猜得到，这两个事件都是由Spring发布的。

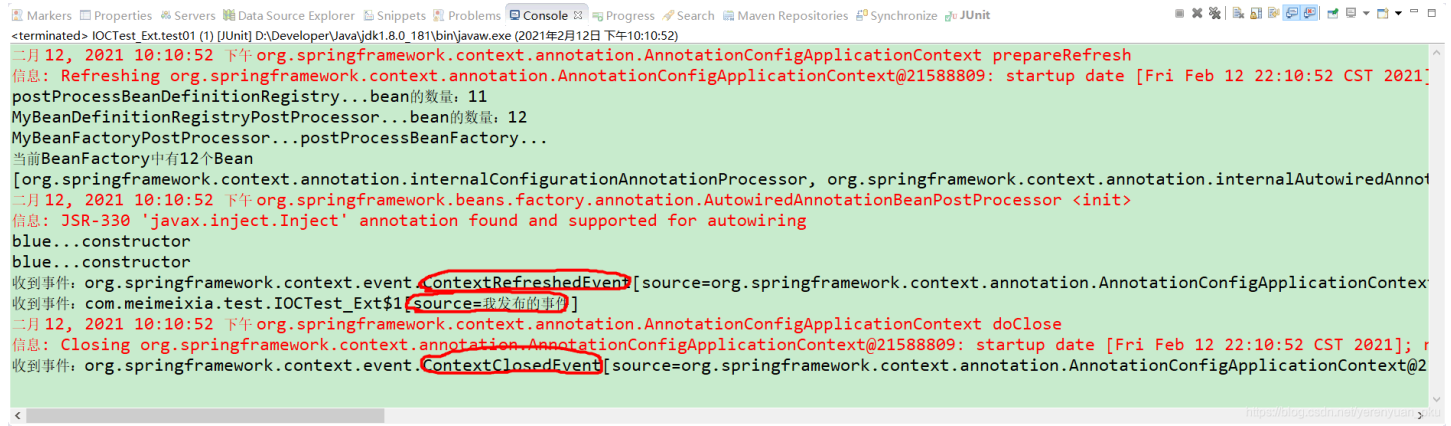
- ContextRefreshEvent：容器刷新完成事件。即容器刷新完成（此时，所有bean都已完全创建），便会发布该事件。
- ContextClosedEvent：容器关闭事件。即容器关闭时，便会发布该事件。

其实，在上面我们也看到了，Spring还默认定义了一些其他事件。除此之外，我们自己也可以编写一些自定义事件。但是，问题的关键是我们能不能自己发布事件呢？答案是可以。

第四步，我们自己来发布一个事件。而发布一个事件，我们需要像下面这么来做。



此时，运行以上test01方法，你将会看到Eclipse控制台打印出了如下内容。



```
<terminated> IOCTest_Ext.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2021年2月12日 下午10:10:52)
二月 12, 2021 10:10:52 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepareRefresh
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@21588809: startup date [Fri Feb 12 22:10:52 CST 2021]; root of context hierarchy
postProcessBeanDefinitionRegistry...bean的数量: 11
MyBeanDefinitionRegistryPostProcessor...bean的数量: 12
MyBeanFactoryPostProcessor...postProcessBeanFactory...
当前BeanFactory中有12个Bean
[org.springframework.context.annotation.internalConfigurationAnnotationProcessor, org.springframework.context.annotation.internalAutowiredAnnotationProcessor]
二月 12, 2021 10:10:52 下午 org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor <init>
信息: JSR-330 'javax.inject.Inject' annotation found and supported for autowiring
blue...constructor
blue...constructor
收到事件: org.springframework.context.event.ContextRefreshedEvent[source=org.springframework.context.annotation.AnnotationConfigApplicationContext@21588809]
收到事件: com.meimeixia.test.IOCTest_Ext$1[source=我发布的事件]
二月 12, 2021 10:10:52 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext doClose
信息: Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@21588809: startup date [Fri Feb 12 22:10:52 CST 2021]; root of context hierarchy
收到事件: org.springframework.context.event.ContextClosedEvent[source=org.springframework.context.annotation.AnnotationConfigApplicationContext@21588809]
```

除了能收到容器刷新完成和容器关闭这两事件之外，还能收到我们调用applicationContext发布出去的事件。只要把这个事件发布出去，那么我们自己编写的监听器就能监听到这个事件。

以上就是ApplicationListener应用监听器的使用。那么，ApplicationListener到底是怎么工作的呢？我们下一讲就来讲讲它内部的原理。