链滴

SpringApplicationRunListener 是如何工作的（旧文迁移）

作者：DeeWooo

原文链接：https://ld246.com/article/1574247667574

来源网站：链滴

2018年7月3日

Springboot启动过程中，使用了SpringApplicationRunListener监听器来监听启动过程的运行情况那么它是如何工作的呢?

Spring在这里设计得很精巧。使用的是观察者模式，可以无需对启动时的其它业务bean的配置关心只需要正常启动创建Spring应用上下文环境。。将SpringApplicationRunListeners作为信息发布者初始化过程中通过META_INF/spring.factories文件中指定的监听器实例化出来的各个SpringApplicationRunListener作为信息订阅者。

各个业务'监听观察者'在监听到spring开始启动，或环境准备完成等事件后，会按照自己的逻辑创建需的bean或者进行相应的配置。观察者模式使run方法的结构变得清晰，同时与外部耦合降到最低。

SpringApplicationEvent事件

在启动Spring Boot时所发生的事件一般指：

- 开始启动事件（ApplicationStartedEvent）
- 环境准备完成事件（ApplicationEnvironmentPreparedEvent）
- 上下文准备完成事件（无）
- 上下文加载完成（ApplicationPreparedEvent）
- 应用启动完成事件（ApplicationReadyEvent或者ApplicationFailedEvent）

Spring启动时的每个事件都是继承自 SpringApplicationEvent 抽象类的一个实现，每一个事件都包了应用的 SpringApplication 对象和应用程序启动时的参数。

SpringApplicationRunListener 监听器

```
/**
 * Listener for the {@link SpringApplication} {@code run} method.
 * {@link SpringApplicationRunListener}s are loaded via the {@link SpringFactoriesLoader}
 * and should declare a public constructor that accepts a {@link SpringApplication}
 * instance and a {@code String[]} of arguments. A new
 * {@link SpringApplicationRunListener} instance will be created for each run.
 *
 * @author Phillip Webb
 * @author Dave Syer
 */
public interface SpringApplicationRunListener {
   /**
    * 在run方法业务逻辑执行、应用上下文初始化前调用此方法
    */
   void starting();
   /**
    * 当环境准备完成，应用上下文被创建之前调用此方法
    */
   void environmentPrepared(ConfigurableEnvironment environment);
   /**
    * 在应用上下文被创建和准备完成之后，但上下文相关代码被加载执行之前调用。因为上下文准备件和上下文加载事件难以明确区分，所以这个方法一般没有具体实现。
    */
   void contextPrepared(ConfigurableApplicationContext context);
   /**
```

```
     * 当上下文加载完成之后，自定义bean完全加载完成之前调用此方法。
     */
    void contextLoaded(ConfigurableApplicationContext context);
    /**
     * 当run方法执行完成，或执行过程中发现异常时调用此方法。
     */
    void finished(ConfigurableApplicationContext context, Throwable exception);
}
```

Spring Boot在启动过程中会实例化EventPublishingRunListener作为SpringApplicationRunListener实例。在实例化监听器时需要SpringApplication对象和用户对象作为参数。其内部维护着一个事件播器（被观察者对象集合，前面所提到的在META_INF/spring.factories中注册的初始化监听器的有集合），当监听到Spring启动等事件发生后，就会将创建具体事件对象，并广播推送给各个被观察。

###SimpleApplicationEventMulticaster事件广播

EventPublishingRunListener中采用SimpleApplicationEventMulticaster作为事件广播器。SimpleApplicationEventMulticaster是ApplicationEventMulticaster的一个实现。

EventPublishingRunListener初始化时，将作为参数传入的SpringApplication中的所有listener放入播器中，供各个方法使用。广播代码如下：

```
//将事件广播出去
@Override
public void multicastEvent(final ApplicationEvent event, ResolvableType eventType) {
    ResolvableType type = (eventType != null ? eventType : resolveDefaultEventType(event));
    for (final ApplicationListener<?> listener : getApplicationListeners(event, type)) {
        Executor executor = getTaskExecutor();
        if (executor != null) {
            executor.execute(new Runnable() {
                @Override
                public void run() {
                    invokeListener(listener, event);
                }
            });
        }
        else {
            invokeListener(listener, event);
        }
    }
}
/**
     * 把指定事件交给指定监听器.
     */
    protected void invokeListener(ApplicationListener<?> listener, ApplicationEvent event) {
        ErrorHandler errorHandler = getErrorHandler();
        if (errorHandler != null) {
            try {
                doInvokeListener(listener, event);
            }
            catch (Throwable err) {
                errorHandler.handleError(err);
            }
        }
```

```
  else {
   doInvokeListener(listener, event);
  }
 }
}
@SuppressWarnings({"unchecked", "rawtypes"})
 private void doInvokeListener(ApplicationListener listener, ApplicationEvent event) {
  try {
   listener.onApplicationEvent(event);
  }
  catch (ClassCastException ex) {
   String msg = ex.getMessage();
   if (msg == null || msg.startsWith(event.getClass().getName())) {
    // Possibly a lambda-defined listener which we could not resolve the generic event type for
    Log logger = LogFactory.getLog(getClass());
    if (logger.isDebugEnabled()) {
     logger.debug("Non-matching event type for listener: " + listener, ex);
    }
   }
   else {
    throw ex;
   }
  }
}
```

最终调用ApplicationListener的onApplicationEvent方法对事件做出处理。