SpringBootSpringBoot中的事件機制

在這篇文章中聊一聊 Spring 中的擴展機制（一）中對 Spring 中的事件機制進行了分析。那麼對於 SpringBoot 來說，它在 Spring 的基礎上又做了哪些拓展呢？本篇將來聊一聊 SpringBoot 中的事件。

在 SpringBoot 的啟動過程中，會通過 SPI 機制去加載 spring.factories 下面的一些類，這裡面就包括了事件相關的類。

SpringApplicationRunListener

```
# Run Listeners
org.springframework.boot.SpringApplicationRunListener=\
org.springframework.boot.context.event.EventPublishingRunListener
```

ApplicationListener

```
# Application Listeners
org.springframework.context.ApplicationListener=\
org.springframework.boot.ClearCachesApplicationListener,\
org.springframework.boot.builder.ParentContextCloserApplicationListener,\
org.springframework.boot.context.FileEncodingApplicationListener,\
org.springframework.boot.context.config.AnsiOutputApplicationListener,\
org.springframework.boot.context.config.ConfigFileApplicationListener,\
org.springframework.boot.context.config.DelegatingApplicationListener,\
org.springframework.boot.context.logging.ClasspathLoggingApplicationListener,\
org.springframework.boot.context.logging.LoggingApplicationListener,\
org.springframework.boot.liquibase.LiquibaseServiceLocatorApplicationListener
```

SpringApplicationRunListener 類是 SpringBoot 中新增的類。SpringApplication 類 中使用它們來間接調用 ApplicationListener 。另外還有一個新增的類是 SpringApplicationRunListeners ， SpringApplicationRunListeners 中包含了多個 SpringApplicationRunListener 。

目錄

SpringApplicationRunListener

SpringApplicationRunListener 接口規定了 SpringBoot 的生命週期，在各個生命週期廣播相應的事件，調用實際的 ApplicationListener 類。通過對 SpringApplicationRunListener 的分析，也可以對 SpringBoot 的整個啟動過程的理解會有很大幫助。

先來看下 SpringApplicationRunListener 接口的代碼：

```
public interface SpringApplicationRunListener {

void starting();

void environmentPrepared(ConfigurableEnvironment environment);

void contextPrepared(ConfigurableApplicationContext context);

void contextLoaded(ConfigurableApplicationContext context);

void started(ConfigurableApplicationContext context);


void running(ConfigurableApplicationContext context);

void failed(ConfigurableApplicationContext context, Throwable exception);
}
```

SpringApplicationRunListeners

上面提到，`SpringApplicationRunListeners` 是 `SpringApplicationRunListener` 的集合，裡面包括了很多 `SpringApplicationRunListener` 實例；`SpringApplication` 類實際上使用的是 `SpringApplicationRunListeners` 類，與 `SpringApplicationRunListener` 生命週期相同，調用各個週期的 `SpringApplicationRunListener` 。然後廣播相應的事件到 `ApplicationListener` 。

> 代碼詳見：SpringApplicationRunListeners.

EventPublishingRunListener

`EventPublishingRunListener` 類是 `SpringApplicationRunListener` 接口的實現類，它具有廣播事件的功能。其內部使用 `ApplicationEventMulticaster` 在實際刷新上下文之前發佈事件。下面來看下 `EventPublishingRunListener` 類生命週期對應的事件。

ApplicationStartingEvent

`ApplicationStartingEvent` 是 `SpringBoot` 啟動開始的時候執行的事件，在該事件中可以獲取到 `SpringApplication` 對象，可做一些執行前的設置，對應的調用方法是 `starting()` 。

ApplicationEnvironmentPreparedEvent

`ApplicationEnvironmentPreparedEvent` 是 `SpringBoot` 對應 `Enviroment` 已經準備完畢時執行的事件，此時上下文 `context` 還沒有創建。在該監聽中獲取到 `ConfigurableEnvironment` 後可以對配置信息做操作，例如：修改默認的配置信息，增加額外的配置信息等。對應的生命週期方法是 `environmentPrepared(environment)` ；`SpringCloud` 中，引導上下文就是在這時初始化的。

ApplicationContextInitializedEvent

當 SpringApplication 啟動並且準備好 ApplicationContext，並且在加載任何 bean 定義之前調用了 ApplicationContextInitializers 時發佈的事件。對應的生命週期方法是 contextPrepared()

ApplicationPreparedEvent

ApplicationPreparedEvent 是 SpringBoot 上下文 context 創建完成是發佈的事件；但此時 spring 中的 bean 還沒有完全加載完成。這裡可以將上下文傳遞出去做一些額外的操作。但是在該監聽器中是無法獲取自定義 bean 並進行操作的。對應的生命週期方法是 contextLoaded()。

ApplicationStartedEvent

這個事件是在 2.0 版本才引入的；具體發佈是在應用程序上下文刷新之後，調用任何 ApplicationRunner 和 CommandLineRunner 運行程序之前。

ApplicationReadyEvent

這個和 ApplicationStartedEvent 很類似，也是在應用程序上下文刷新之後之後調用，區別在於此時 ApplicationRunner 和 CommandLineRunner 已經完成調用了，也意味著 SpringBoot 加載已經完成。

ApplicationFailedEvent

SpringBoot 啟動異常時執行的事件，在異常發生時，最好是添加虛擬機對應的鉤子進行資源的回收與釋放，能友善的處理異常信息。

demo 及各個事件的執行順序

下面的各個事件對應的demo及打印出來的執行順序。

GlmapperApplicationStartingEventListener

```
public class GlmapperApplicationStartingEventListener implements
ApplicationListener<ApplicationStartingEvent> {
@Override
public void onApplicationEvent(ApplicationStartingEvent applicationStartingEvent) {
System.out.println("execute ApplicationStartingEvent ...");
}
}
```

GlmapperApplicationEnvironmentPreparedEvent

```
public class GlmapperApplicationEnvironmentPreparedEvent implements
ApplicationListener<ApplicationEnvironmentPreparedEvent> {
@Override
public void onApplicationEvent(ApplicationEnvironmentPreparedEvent
applicationEnvironmentPreparedEvent) {
System.out.println("execute ApplicationEnvironmentPreparedEvent ...");
}
}
```

GlmapperApplicationContextInitializedEvent

```
public class GlmapperApplicationContextInitializedEvent implements
ApplicationListener<ApplicationContextInitializedEvent> {
@Override
public void onApplicationEvent(ApplicationContextInitializedEvent applicationContextInitializedEvent)
{
System.out.println("execute applicationContextInitializedEvent ...");
}
}
```

GlmapperApplicationPreparedEvent

```
public class GlmapperApplicationPreparedEvent implements
ApplicationListener<ApplicationPreparedEvent> {
@Override
public void onApplicationEvent(ApplicationPreparedEvent applicationPreparedEvent) {
System.out.println("execute ApplicationPreparedEvent ...");
}
}
```

GlmapperApplicationStartedEvent

```
public class GlmapperApplicationStartedEvent implements
ApplicationListener<ApplicationStartedEvent> {
@Override
public void onApplicationEvent(ApplicationStartedEvent applicationStartedEvent) {
System.out.println("execute ApplicationStartedEvent ...");
}
}
```

GlmapperApplicationReadyEvent

```
public class GlmapperApplicationReadyEvent implements
ApplicationListener<ApplicationReadyEvent> {
@Override
public void onApplicationEvent(ApplicationReadyEvent applicationReadyEvent) {
System.out.println("execute ApplicationReadyEvent ...");
}
}
```

執行結果

SpringBoot 中的事件體系

這裡圍繞 `SpringApplicationRunListener` 這個類來說。在實現類 `EventPublishingRunListener` 中，事件發佈有兩種模式：

- 通過 `SimpleApplicationEventMulticaster` 進行事件廣播
- 所有監聽器交給相應的 `Context`

所以 `EventPublishingRunListener` 不僅負責發佈事件，而且在合適的時機將 `SpringApplication` 所獲取的監聽器和應用上下文作關聯。

SimpleApplicationEventMulticaster

`SimpleApplicationEventMulticaster` 是 `Spring` 默認的事件廣播器。來看下它是怎麼工作的：

```
@Override
public void multicastEvent(final ApplicationEvent event, @Nullable ResolvableType eventType) {
ResolvableType type = (eventType != null ? eventType : resolveDefaultEventType(event));
for (final ApplicationListener<?> listener : getApplicationListeners(event, type)) {
Executor executor = getTaskExecutor();
if (executor != null) {
executor.execute(() -> invokeListener(listener, event));
}
else {
invokeListener(listener, event);
}
}
}
```

從上面的代碼段可以看出，它是通過遍歷註冊的每個監聽器，並啟動來調用每個監聽器的 `onApplicationEvent` 方法。

下面再來看下 `SimpleApplicationEventMulticaster` 的類集成結構：

這裡的 `AbstractApplicationContext` 下面來聊，這個類實際上就負責了事件體系的初始化工作。

事件體系的初始化

事件體系的初始化對應在 `SpringBoot` 啟動過程的 `refreshContext` 這個方法； `refreshContext` 具體調用 AbstractApplicationContext.refresh()方法，最後調用 initApplicationEventMulticaster() 來完成事件體系的初始化,代碼如下：

用戶可以為容器定義一個自定義的事件廣播器，只要實現 `ApplicationEventMulticaster` 就可以了， `Spring` 會通過 反射的機制將其註冊成容器的事件廣播器，如果沒有找到配置的外部事件廣播器， `Spring` 就是默認使用 `SimpleApplicationEventMulticaster` 作為事件廣播器。

事件註冊

事件註冊是在事件體系初始化完成之後做的事情，也是在 `AbstractApplicationContext.refresh()` 方法中進行調用的。

這裡幹了三件事：

- 首先註冊靜態指定的 `listeners` ；這裡包括我們自定義的那些監聽器。
- 調用 `DefaultListableBeanFactory` 中 `getBeanNamesForType` 得到自定義的 `ApplicationListener` `bean` 進行事件註冊。
- 廣播早期的事件。

事件廣播

事件發佈伴隨著 `SpringBoot` 啟動的整個生命週期。不同階段對應發佈不同的事件，上面我們已經對各個事件進行了分析，下面就具體看下發布事件的實現：

> org.springframework.context.support.AbstractApplicationContext#publishEvent

> earlyApplicationEvents 中的事件是廣播器未建立的時候保存通知信息，一旦容器建立完成，以後都是直接通知。

廣播事件最終還是通過調用 `ApplicationEventMulticaster` 的 `multicastEvent` 來實現。而 `multicastEvent` 也就就是事件執行的方法。

事件執行

上面 `SimpleApplicationEventMulticaster` 小節已經初步介紹了 `multicastEvent` 這個方法。補充一點， 如果有可用的 `taskExecutor` 會使用併發的模式執行事件，但是實際上 `SimpleApplicationEventMulticaster` 並沒有提供線程池實現，默認請況下是使用同步的方式執行事件（ `org.springframework.core.task.SyncTaskExecutor` ），所以如果需要異步配置的話，需要自己去實現線程池。

SpringBoot 啟動過程中的事件階段

這裡回到 `SpringApplication` 的 `run` 方法，看下 `SpringBoot` 在啟動過程中，各個事件階段做了哪些事情。

starting -> ApplicationStartingEvent

這裡 `debug` 到 `starting` 方法，追蹤到 `multicastEvent` ，這裡 `type` 為 `ApplicationStartingEvent` ；對應的事件如下：

- LoggerApplicationListener：配置日誌系統。使用 `logging.config` 環境變量指定的配置或者缺省配置
- BackgroundPreinitializer：儘早觸發一些耗時的初始化任務，使用一個後臺線程
- DelegatingApplicationListener：監聽到事件後轉發給環境變量 `context.listener.classes` 指定的那些事件監聽器
- LiquibaseServiceLocatorApplicationListener：使用一個可以和 `SpringBoot` 可執行 `jar` 包配合工作的版本替換 `liquibase ServiceLocator`

listeners.environmentPrepared->ApplicationEnvironmentPreparedEvent

- AnsiOutputApplicationListener：根據 `spring.output.ansi.enabled` 參數配置 `AnsiOutput`

- ConfigFileApplicationListener： `EnvironmentPostProcessor`，從常見的那些約定的位置讀取配置文件，比如從以下目錄讀取 `application.properties`，`application.yml` 等配置文件：

  - classpath:
  - file:.
  - classpath:config
  - file:./config/

  也可以配置成從其他指定的位置讀取配置文件。

- ClasspathLoggingApplicationListener：對環境就緒事件 `ApplicationEnvironmentPreparedEvent` /應用失敗事 件`ApplicationFailedEvent` 做出響應，往日誌 `DEBUG` 級別輸出 `TCCL(thread context class loader)` 的 `classpath`。

- FileEncodingApplicationListener：如果系統文件編碼和環境變量中指定的不同則終止應用啟動。具體的方法是比較系統屬性 `file.encoding` 和環境變量 `spring.mandatory-file-encoding` 是否相等(大小寫不敏感)。

listeners.contextPrepared->ApplicationContextInitializedEvent

相關監聽器參考上面的描述。

listeners.contextLoaded->ApplicationPreparedEvent

相關監聽器參考上面的描述。

refresh->ContextRefreshedEvent

- ConditionEvaluationReportLoggingListener：實際上實現的是 `ApplicationContextInitializer` 接口，其目的是將 `ConditionEvaluationReport` 寫入到日誌，使用 `DEBUG` 級別輸出。程序崩潰報告會觸發一個消息輸出，建議用戶使用調試模式顯示報告。它是在應用初始化時綁定一個 `ConditionEvaluationReportListener` 事件監聽器，然後相應的事件發生時輸出 `ConditionEvaluationReport` 報告。
- ClearCachesApplicationListener：應用上下文加載完成後對緩存做清除工作，響應事件 `ContextRefreshedEvent`。
- SharedMetadataReaderFactoryContextInitializer： 向 `context` 註冊了一個 `BeanFactoryPostProcessor`：`CachingMetadataReaderFactoryPostProcessor` 實例。
- ResourceUrlProvider：`handling mappings` 處理

started->ApplicationStartedEvent

相關監聽器參考上面的描述。

running->ApplicationReadyEvent

相關監聽器參考上面的描述。

BackgroundPreinitializer&DelegatingApplicationListener

這兩個貫穿了整個過程，這裡拎出來單獨解釋下：

- BackgroundPreinitializer：對於一些耗時的任務使用一個後臺線程儘早觸發它們開始執行初始化，這是 `SpringBoot` 的缺省行為。這些初始化動作也可以叫做預初始化。可以通過設置系統屬性 `spring.backgroundpreinitializer.ignore` 為 `true` 可以禁用該機制。該機制被禁用時，相應的初始化任務會發生在前臺線程。
- DelegatingApplicationListener：監聽應用事件，並將這些應用事件廣播給環境屬性 `context.listener.classes` 指定的那些監聽器。

小結

到此，`SpringBoot` 中的事件相關的東西就結束了。本文從 `SpringApplicationRunListener` 這個類說起，接著介紹 `SpringBoot` 啟動過程的事件以及事件的生命週期。最後介紹了 `SpringBoot` 中的內置的這些 監聽器在啟動過程中對應的各個階段。