Spring 的核心是 ApplicationContext，它负责管理 beans 的完整生命周期。当加载 beans 时，ApplicationContext 发布某些类型的事件。eg.当上下文启动时，ContextStartedEvent 发布。当上下文停止时，ContextStoppedEvent 发布。通过 ApplicationEvent 类和 ApplicationListener 接口来处理事件。如果一个 bean 实现 ApplicationListener，那么每次 ApplicationEvent 被发布到 ApplicationContext 上，该bean 会被通知。

标准事件：

序号	Spring 内置事件 & 描述
1	**ContextRefreshedEvent** ApplicationContext 被初始化或刷新时，该事件被发布。这也可以在 ConfigurableApplicationContext 接口中使用 refresh() 方法来发生。
2	**ContextStartedEvent** 当使用 ConfigurableApplicationContext 接口中的 start() 方法启动 ApplicationContext 时，该事件被发布。你可以调查你的数据库，或者你可以在接受到这个事件后重启任何停止的应用程序。
3	**ContextStoppedEvent** 当使用 ConfigurableApplicationContext 接口中的 stop() 方法停止 ApplicationContext 时，发布这个事件。你可以在接受到这个事件后做必要的清理的工作。
4	**ContextClosedEvent** 当使用 ConfigurableApplicationContext 接口中的 close() 方法关闭 ApplicationContext 时，该事件被发布。一个已关闭的上下文到达生命周期末端；它不能被刷新或重启。
5	**RequestHandledEvent** 这是一个 web-specific 事件，告诉所有 bean HTTP 请求已经被服务。

由于 Spring 的事件处理是单线程的，所以如果一个事件被发布，直至并且除非所有的接收者得到该消息，否则该进程被阻塞并且流程将不会继续。

监听上下文事件

一个bean应该实现只有一个方法 onApplicationEvent()的 ApplicationListener 接口。例子：

HelloWorld.java ：

```
package com.tutorialspoint;
public class HelloWorld {
   private String message;
   public void setMessage(String message){
      this.message  = message;
   }
   public void getMessage(){
      System.out.println("Your Message : " + message);
   }
}
```

CStartEventHandler.java：

```
package com.tutorialspoint;
import org.springframework.context.ApplicationListener;
import org.springframework.context.event.ContextStartedEvent;
public class CStartEventHandler
   implements ApplicationListener<ContextStartedEvent>{
   public void onApplicationEvent(ContextStartedEvent event) {
      System.out.println("ContextStartedEvent Received");
   }
}
```

CStopEventHandler.java ：

```
package com.tutorialspoint;
import org.springframework.context.ApplicationListener;
import org.springframework.context.event.ContextStoppedEvent;
public class CStopEventHandler
   implements ApplicationListener<ContextStoppedEvent>{
   public void onApplicationEvent(ContextStoppedEvent event) {
      System.out.println("ContextStoppedEvent Received");
   }
}
```

MainApp.java：

```
package com.tutorialspoint;
```

```
import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ConfigurableApplicationContext context =
      new ClassPathXmlApplicationContext("Beans.xml");

      // Let us raise a start event.
      context.start();

      HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

      obj.getMessage();

      // Let us raise a stop event.
      context.stop();
   }
}
```

Beans.xml ：

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

   <bean id="helloWorld" class="com.tutorialspoint.HelloWorld">
      <property name="message" value="Hello World!"/>
   </bean>

   <bean id="cStartEventHandler"
         class="com.tutorialspoint.CStartEventHandler"/>

   <bean id="cStopEventHandler"
         class="com.tutorialspoint.CStopEventHandler"/>
```

```
</beans>
```

运行：

```
ContextStartedEvent Received
Your Message : Hello World!
ContextStoppedEvent Received
```

自定义事件

CustomEvent.java ：

```
package com.tutorialspoint;
import org.springframework.context.ApplicationEvent;
public class CustomEvent extends ApplicationEvent{
   public CustomEvent(Object source) {
      super(source);
   }
   public String toString(){
      return "My Custom Event";
   }
}
```

CustomEventPublisher.java ：

```
package com.tutorialspoint;
import org.springframework.context.ApplicationEventPublisher;
import org.springframework.context.ApplicationEventPublisherAware;
public class CustomEventPublisher
   implements ApplicationEventPublisherAware {
   private ApplicationEventPublisher publisher;
   public void setApplicationEventPublisher
               (ApplicationEventPublisher publisher){
      this.publisher = publisher;
   }
   public void publish() {
      CustomEvent ce = new CustomEvent(this);
      publisher.publishEvent(ce);
   }
}
```

下面是 CustomEventHandler.java 文件的内容：

```
package com.tutorialspoint;
import org.springframework.context.ApplicationListener;
public class CustomEventHandler
   implements ApplicationListener<CustomEvent>{
   public void onApplicationEvent(CustomEvent event) {
      System.out.println(event.toString());
   }
}
```

下面是 MainApp.java 文件的内容：

```
package com.tutorialspoint;
import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class MainApp {
   public static void main(String[] args) {
      ConfigurableApplicationContext context =
      new ClassPathXmlApplicationContext("Beans.xml");
      CustomEventPublisher cvp =
      (CustomEventPublisher) context.getBean("customEventPublisher");
      cvp.publish();
      cvp.publish();
   }
}
```

下面是配置文件 Beans.xml：

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

   <bean id="customEventHandler"
      class="com.tutorialspoint.CustomEventHandler"/>

   <bean id="customEventPublisher"
```

```
      class="com.tutorialspoint.CustomEventPublisher"/>

</beans>
```

一旦你完成了创建源和 bean 的配置文件后，我们就可以运行该应用程序。如果你的应用程序一切都正常，将输出以下信息：

```
My Custom Event
My Custom Event
```