

ApplicationContext Event Yönetimi



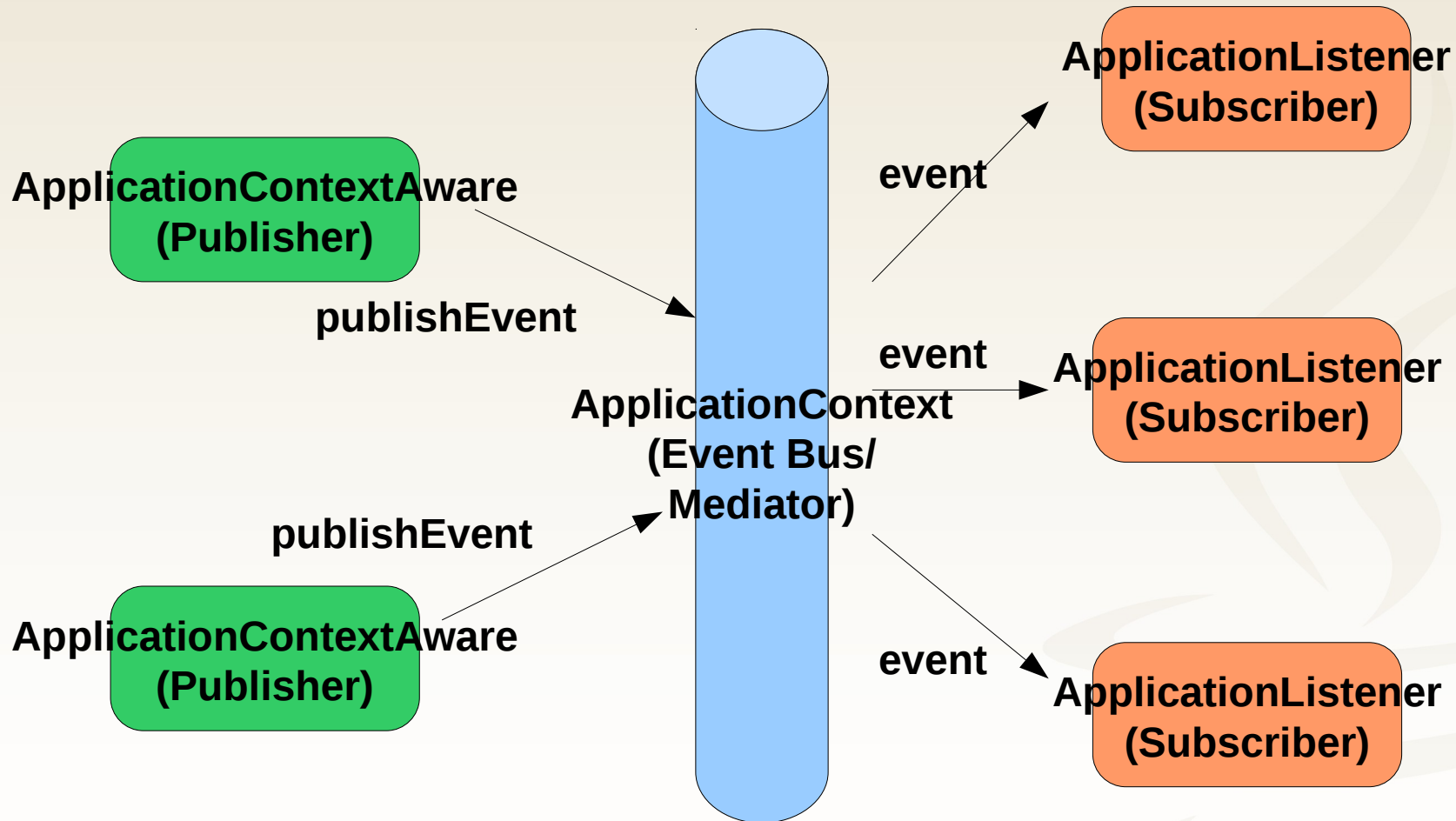
ApplicationContext ve Event'ler

- ApplicationContext üzerinden belirli durumlarda **event'ler fırlatılabilir**
- Fırlatılan event'ler **ApplicationEvent** tipinde nesnelerdir
- Event fırlatma işi
ApplicationContext.**publishEvent** metodu ile gerçekleştirilir
- Event fırlatacak Spring managed bean'lerine **ApplicationContext enjekte edilmelidir**

ApplicationEvent'lerin Yakalanması

- Diğer Spring managed bean'ları da bu **event'leri yakalayıp** birtakım işlemler gerçekleştirebilirler
- Event handler bean sınıflarının ise **ApplicationListener** arayüzünü implement etmesi gerekir

ApplicationContext Event Mimarisi



Burada **Observer** ve **Mediator** örüntüleri birlikte kullanılmaktadır

ApplicationContext'de Event Yönetimi

- ApplicationListener nesnelerine event'ler **senkron** biçimde verilir
- Bu durumda publishEvent() bütün listener'lar çalışana değin **süreci bloklar**
- İstenirse **ApplicationEventMulticaster** arayüzü implement edilerek event yönetimi **asenkron biçimde** de gerçekleştirilebilir

Built-in Event Tipleri

- Spring Container **built-in event'ler** fırlatmaktadır
 - ContextRefreshedEvent
 - ContextClosedEvent
 - RequestHandledEvent
- Ayrıca Spring Security gibi diğer framework'lerin de **built-in event'leri** mevcuttur
 - AuthenticationSuccessEvent
 - AbstractAuthenticationFailureEvent

Uygulamaya Özel Event Tipleri

- Uygulamaya özel event'ler de tanımlanıp, ApplicationContext üzerinden fırlatılabilir
- Event sınıfları **ApplicationEvent** sınıfından türetilmelidir
- Bu event'leri yaratıp fırlatmak için de **ApplicationContext.publishEvent()** metodu kullanılır

Uygulamaya Özel Event Tipleri

```
public class EmailSender implements ApplicationContextAware {  
  
    private List blackList;  
    private ApplicationContext appContext;  
  
    public void sendEmail(String address, String text) {  
        if (blackList.contains(address)) {  
            BlackListEvent event = new BlackListEvent(address, text);  
            appContext.publishEvent(event);  
            return;  
        }  
        // email gönderme işlemi...  
    }  
}
```


Uygulamaya Özel Event Tipleri

```
public class BlackListNotificationHandler
    implements ApplicationListener {

    private String notificationAddress;

    ...

    public void onApplicationEvent(ApplicationEvent
event) {
        if (event instanceof BlackListEvent) {
            //sistem yoneticisi vs. bilgilendirilir...
        }
    }
}
```

Uygulamaya Özel Event Tipleri (Java Generics)

```
public class BlackListNotificationHandler
    implements ApplicationListener<BlackListEvent> {

    private String notificationAddress;

    ...

    public void onApplicationEvent(BlackListEvent event) {
        //sistem yoneticisi vs. bilgilendirilir...
    }
}
```

Uygulamaya Özel Event Tipleri

```
<bean id="emailSender" class="example.EmailSender">  
  <property name="blackList">  
    <list>  
      <value>black@list.org</value>  
      <value>white@list.org</value>  
      <value>john@doe.org</value>  
    </list>  
  </property>  
</bean>
```

```
<bean id="blackListListener"  
class="example.BlackListNotificationHandler">  
  
  <property name="notificationAddress" value="spam@list.org"/>  
  
</bean>
```

Annotasyon Tabanlı Event Listener Tanımı

- Spring 4.2 ile birlikte **ApplicationListener** arayüzünü **implement etmeden** de event handler tanımlamak mümkün hale gelmiştir
- Bunun için **metot düzeyinde @EventListener anotasyonu** kullanılır
- Bu sayede bir bean içerisinde **birden fazla farklı türde event'i yakalayan metotlar** yazmak mümkündür

Annotasyon Tabanlı Event Listener Tanımı

```
@Component
public class MyApplicationEventListener {

    @EventListener
    public void handle(ContextRefreshedEvent event) {
        System.out.println("ApplicationContext is ready to serve...");
    }

    @EventListener
    public void handle(ContextClosedEvent event) {
        System.out.println("ApplicationContext shutdown...");
    }
}
```


Transactional Event Listener Kabiliyeti

- **@TransactionalEventListener** anotasyonu event'lerin sadece **transactional bir metot içerisinde** fırlatıldıklarında yakalanmalarını sağlar
- Eğer event transactional bir metot içerisinde fırlatılmamış ise **göz ardı** edilir
- Metodun çağırılma zamanı da **TransactionPhase** ile belirlenebilir
- Böylece metodun TX **commit öncesi** veya **sonrası**, yada sadece **rollback** olduğunda invoke edilmesi sağlanabilir

Transactional Event Listener Kabiliyeti

```
@Component
public class MyApplicationEventListener {

    @TransactionalEventListener(
        phase=TransactionPhase.AFTER_COMMIT)
    public void handle(PetCreatedEvent event) {
        //handle pet created event here...
    }
}
```



BEFORE_COMMIT
AFTER_COMMIT
AFTER_ROLLBACK
AFTER_COMPLETION

Ayrıca **fallbackExecution=true** ifadesi ile transactional context dışında fırlatılan bir event'in de yakalanması sağlanabilir

İletişim

- **Harezmi** Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- <http://www.java-egitimleri.com>
- info@java-egitimleri.com

