

# İleri Düzey JPA/Hibernate Eğitimi 5



# Hibernate ve JPA'da Event Sistemi

- Hibernate ve JPA, **persistence işlemleri ile ilgili event'ler** fırlatırlar
- Uygun handler implemantasyonları ile bu **event'ler yakalanarak uygulamaya özel işlemler** gerçekleştirilebilir
- Hibernate ve JPA'nın gelişim sürecinde **farklı türde event mekanizmaları** ortaya çıkmıştır

# Hibernate 3 Öncesi Dönem ve Interceptor Arayüzü

- Hibernate 3 öncesi dönemde Session üzerinde meydana gelen işlemleri intercept etmek için kullanılan yöntemdir
- **Interceptor** arayüzü implement edilerek Hibernate event'lerinde işlem yapılabilir
- **EmptyInterceptor** default implementasyondur
- Interceptor'ü aktive etmek için **Configuration.setInterceptor()** ile interceptor nesnesi set edilmelidir
- Bu işlem SessionFactory **build edilmeden** yapılmalıdır

# Hibernate Interceptor Konfigürasyonu

- Interceptor konfigürasyonu **Configuration** nesnesi üzerinden programatik yapılır
- Configuration düzeyinde tanım **SessionFactory genelinde aktif** olacaktır
- Dolayısı ile bu Interceptor **bütün Session'larda devreye girecektir**

```
Configuration cfg = new Configuration();  
  
cfg.setInterceptor(new AuditLogInterceptor());  
  
cfg.configure();  
SessionFactory sessionFactory = cfg.buildSessionFactory();
```

# Hibernate Interceptor Konfigürasyonu ve JPA

```
<persistence-unit name="jpa-hibernate">  
  <properties>  
    <property name="hibernate.ejb.interceptor"  
value="com.javaegitimleri.AuditLogInterceptor"/>  
    ...  
  </properties>  
</persistence-unit>
```

↓  
Hibernate Interceptor **JPA** konfigürasyonunda  
**EntityManagerFactory/SessionFactory** düzeyinde  
tanıtılabilir

# Hibernate Interceptor Konfigürasyonu ve JPA

```
<persistence-unit name="jpa-hibernate">  
  <properties>  
    <property  
name="hibernate.ejb.interceptor.session_scoped"  
value="com.javaegitimleri.AuditLogInterceptor"/>  
    ...  
  </properties>  
</persistence-unit>
```

Hibernate Interceptor **JPA** konfigürasyonunda  
**EntityManager/Session** düzeyinde de tanımlanabilir  
Bu durumda her EntityManager/Session oluşturulduğunda  
bu PersistenceContext'e özgü yeni bir Interceptor  
instance'ı yaratılacaktır

# Hibernate Interceptor API

```
public class AuditLogInterceptor extends EmptyInterceptor {

    public boolean onLoad(Object entity, Serializable id,
        Object[] state, String[] propertyNames, Type[] types) {
        return false;
    }

    public boolean onSave(Object entity, Serializable id,
        Object[] state, String[] propertyNames, Type[] types) {
        return false;
    }

    public boolean onFlushDirty(Object entity, Serializable id,
        Object[] currentState, Object[] previousState,
        String[] propertyNames, Type[] types) {
        return false;
    }

    public void onDelete(Object entity, Serializable id,
        Object[] state, String[] propertyNames, Type[] types) {
    }

    public void afterTransactionCompletion(Transaction tx) {
    }
}
```

State  
üzerinde  
değişiklik  
yapılırsa  
true  
dönülür

# Hibernate Interceptor Kullanımında Dikkat Edilecek Noktalar

- Interceptor içerisinde bir nesnenin **save()** işlemi sırasında aynı **Session** üzerinden yeni bir **save()** metot çağrısı önerilmez
- Böyle bir durumda ikinci save işlemi **farklı bir Session** üzerinden gerçekleştirilmelidir



# Hibernate 3 ve Core Event Sistemi

- Hibernate 3 mimarisi ise **event ve listener modeli** üzerine bina edilmiştir
- **Session** nesnesindeki **hemen her işlem event** tetikler
- Örneğin, bir entity merge edildiğinde **merge event** fırlatılır
- Bazı işlemler **birden çok event** tetikleyebilir
- Örneğin, save işlemi **pre-insert, save ve post-insert** event'lerini tetikler

# Hibernate Session Tarafından Fırlatılan Event'ler

- auto-flush
- merge
- create
- create-onflush
- delete
- dirty-check
- evict
- flush
- flush-entity
- load
- load-collection
- lock
- refresh
- replicate
- save-update
- save
- update
- pre-load
- pre-update
- pre-insert
- pre-delete
- pre-collection-recreate
- pre-collection-remove
- pre-collection-update
- post-load
- post-update
- post-insert
- post-delete
- post-collection-recreate
- post-collection-remove
- post-collection-update
- post-commit-update
- post-commit-insert
- post-commit-delete

# Event Sınıfları ve Listener Arayüzleri

- Her bir event için bir **event sınıfı** ve bir **listener** arayüzü vardır
- Örneğin **save event**'i **SaveEvent** ile ifade edilirken, handle etmek için **SaveOrUpdateEventListener** arayüzü vardır
- Hibernate kabiliyetleri de **bir grup default listener** ile hayata geçirilmiştir

# Custom Event Listener

- Bu listener arayüzlerini implement eden **custom listener** yazmak ve devreye almak mümkündür
- Custom event listener için ya event'e uygun **arayüz implement** edilir yada **default event listener**'lar extend edilebilir

```
public class CustomLoadListener implements LoadEventListener {

    @Override
    public void onLoad(LoadEvent event, LoadType loadType) throws
    HibernateException {
        //load event ile ilgili uygulamaya özel işler burada
        gerçekleştirilir...
    }

}
```

# Event Listener'ların Tanıtılması

- **hibernate.cfg.xml** dosyasında event tipi için custom listener tanıtılır
- Custom listener'ın yanı sıra bu event'le ilgili **default listener'ın da tekrardan tanıtılması** gerekir
- Aksi takdirde Hibernate'in **default davranışı devre dışı** kalacaktır

```
<hibernate-configuration>  
  <session-factory>  
    <event type="load">  
      <listener class="org.hibernate.event.def.DefaultLoadEventListener"/>  
      <listener class="com.javaegitimleri.petclinic.dao.CustomEventListener"/>  
    </event>  
  </session-factory>  
</hibernate-configuration>
```

# Event Listener'ların Tanıtılması: Hibernate 4

- Hibernate 4'de **hibernate.cfg.xml** içindeki event listener tanımları **geçersizdir** (Bu durum Hibernate 5'de düzelmiştir)
- Burada **Java SPI** üzerine kurulu **ServiceRegistry** üzerinden listener tanıtımı yapılabilir
- Önce **org.hibernate.integrator.spi.Integrator** arayüzünü implement eden bir sınıf yazılır
- Bu sınıfın **integrate** metodu içerisinde **EventListenerRegistry**'ye erişilerek custom listener sınıfı default listener'ın yanına eklenebilir

# Event Listener'ların Tanıtılması: Hibernate 4

- Ardından classpath'de **/META-INF/services/org.hibernate.integrator.spi.Integrator** dosyası oluşturularak, içerisine de oluşturulan sınıfın FQN'i yazılır

```
public class EventRegistryIntegrator implements Integrator {  
  
    @Override  
    public void integrate(Configuration configuration,  
        SessionFactoryImplementor sessionFactory,  
        SessionFactoryServiceRegistry serviceRegistry) {  
  
        EventListenerRegistry registry = serviceRegistry  
            .getService(EventListenerRegistry.class);  
  
        registry.appendListeners(EventType.LOAD, CustomLoadListener.class);  
    }  
    ...  
}
```

# Event Listener'ların Tanıtılması: JPA

- Her bir event tipi için karşılık gelen property ismi bu event'in adını son ek alarak belirlenir
- Örneğin load event için **hibernate.ejb.event.load** kullanılır

```
<persistence-unit name="...">
  <properties>
    <property name="hibernate.ejb.event.load"
value="com.javaegitimleri.CustomLoadListener, ..." />
    ...
  </properties>
</persistence-unit>
```



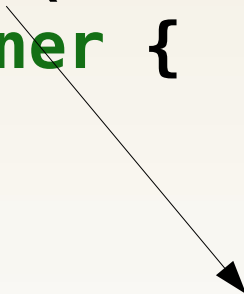
# JPA Persistence Callback Metotları

- JPA ile birlikte persistent işlemler sırasında **callback metotların** devreye girmesi mümkün hale gelmiştir

```
public class OwnerEventListener {  
    @PostLoad  
    private void init(Owner o) {  
    }  
  
    @PreRemove  
    private void cleanup(Owner o) {  
    }  
}
```

# JPA Entity Event Listener ve Callback Metotları

```
@Entity
@EntityListeners(OwnerEventListener.class)
public class Owner {
    //...
}
```



Callback metotları içeren sınıf  
**@EntityListeners** anotasyonu ile tanıtılır.  
Yukarıdaki durumda listener içindeki metotlar  
**sadece Owner entity'lerinin event'lerinde**  
devreye girer

# JPA Persistence Callback Anotasyonları

- **@PostLoad**
  - find(), getReference(), refresh(), veya sorgulama sonrası çalışır
- **@PrePersist, @PostPersist**
  - persist() çağırıldığı vakit hemen ve db insert sonrası çalışır

# JPA Persistence Callback Anotasyonları

- **@PreUpdate, @PostUpdate**
  - Flush öncesi ve sonrası çalışır
  - Sadece entity state senkronize edilirken devreye girer
- **@PreRemove, @PostRemove**
  - remove() metodu çağırıldığında, entity cascade ile silindiği vakit yada, db delete sonrası çağırılır

# Entity İçerisinde Persistence Callback Metotları

@Entity

*public class* Owner {

@PostLoad

*private void* **init**() {

}

@PreRemove

*private void* **cleanup**() {

}

}

İstenirse bu callback metotlar domain sınıfının kendi içinde de yazılabilir  
Aynı kurallar geçerlidir, ancak input argüman verilmez

# Persistence Callback Metotlarında Dikkat Edilecek Noktalar

- Bütün callback metotları **herhangi bir görünürlük düzeyinde** olabilirler,
- Entity nesneyi **input argüman** alabilirler, ve **sadece void** return tipinde olmalıdırlar
- Checked exception **fırlatamazlar**
- Eğer **RuntimeException** fırlatılırsa ve JTA TX'de çalışıyorsa rollback yapılır
- Callback metotları içerisinde **EntityManager** veya **query işlemleri yapılmamalıdır**

# Persistence Callback Metotlarında Dikkat Edilecek Noktalar

- Başka entity'lere erişilmemeli veya ilişkiler değiştirilmemelidir
- Entity'nin ilişkiler dışındaki state'i üzerinde işlem yapılabilir
- Daha önce tanımlanmış belirli bir listener grubunu **exclude** etmek de mümkündür
- @ExcludeSuperclassListeners ve @ExcludeDefaultListeners ile bu yapılabilir

# Konfigürasyon Parametreleri

- **hibernate.max\_fetch\_depth=0-3**
  - 1:1 ve M:1 EAGER ilişkiler için geçerlidir
  - Fetch edilen object graph'ın outer join derinliğini belirler
  - 0 devre dışı bırakır, bu durumda EAGER ilişkiler ayrı SELECT ifadeleri ile yüklenirler



# Konfigürasyon Parametreleri

- **hibernate.default\_batch\_fetch\_size=4,8,16**
  - Hibernate'in lazy ilişkileri batch fetching ile getirmesini sağlar
  - Entity veya ilişki düzeyinde @BatchSize annotasyonu ile fine tuning de yapılabilir

# Konfigürasyon Parametreleri

- **hibernate.jdbc.fetch\_size=100**
  - JDBC driver'da statement'ın fetch size değerini değiştirir
  - Eğer sorgu sonucu birden fazla kayıt dönülüyor ise bu değere göre tek seferde kaç satırın fetch edileceği belirlenir
  - Uygun değer uygulamanın davranışı incelenerek tespit edilebilir
  - Örneğin sorgu sonuçlarında max 20 değer dönülüyorsa 20, 500 değer dönülüyor ise 500 set etmek uygun olacaktır

# Konfigürasyon Parametreleri

- **hibernate.order\_inserts=true**
- **hibernate.order\_updates=true**
- **hibernate.jdbc.batch\_size=(5-30)**
- **hibernate.jdbc.batch\_versioned\_data=true**
  - Bir grup INSERT ve UPDATE ifadesinin tek seferde JDBC API üzerinden DB'ye gönderilmesi sağlanır
  - Hepsi birlikte kullanılmalıdır
  - INSERT işlemlerinin sıralanabilmesi için ID generation stratejisinin uygun olması gerekir (SEQUENCE veya UUID)
  - JDBC driver'ın da desteklemesi gerekir (Oracle desteklemez)
  - Second level cache'in de devre dışı bırakılması gerekir

# Konfigürasyon Parametreleri

- DELETE ifadelerinde ise CascadeType.DELETE nedeni ile JDBC batch kabiliyeti çok efektif olmamaktadır
- JDBC batch işlemi tablo düzeyinde oluşturulur
- Farklı tablolardan ifadeler aynı batch içerisine konamaz
- Cascade özelliği nedeni ile DELETE ifadelerinin kendi aralarında gruplanarak sıralanması mümkün olmamaktadır

# Konfigürasyon Parametreleri

- Bunu aşmak için Cascade özelliğini göz ardı edip Hibernate ile silme işlemini her bir child entity için explicit yapıp, ardından flush yapılabilir
- Bu durumda DELETE ifadeleri uygun biçimde gruplanarak sıralı biçimde topluca çalıştırılacaktır

# Konfigürasyon Parametreleri

- **javax.persistence.validation.mode=NONE**
  - INSERT, DELETE ve UPDATE'ler öncesi yapılan bean validation devre dışı bırakılabilir

# Konfigürasyon Parametreleri

- **hibernate.show\_sql=false**
- **hibernate.format\_sql=false**
- **hibernate.use\_sql\_comments=false**
  - HQL ve SQL logları devre dışı bırakılabilir

# Konfigürasyon Parametreleri

- **hibernate.generate\_statistics=false**
  - Hibernate ile gerçekleştirilen persistence işlemleri ve sorgularla ilgili SessionFactory düzeyindeki istatistik üretimi devre dışı bırakılabilir
  - Default durumda devre dışıdır



# İletişim

- Harezmi Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- <http://www.java-egitimleri.com>
- [info@java-egitimleri.com](mailto:info@java-egitimleri.com)

