

# Hibernate Annotations



Presented by Hosein Zare  
Twitter: @zare88r





# XML vs Annotations

- استفاده کردن از XML فایل ها در نگاشت اشیا به جداول ، مزایا و معایب خود را دارد ، آنها ساده هستند و قابلیت خوانایی بیشتری دارند اما بسیار حجیم هستند و همچنین type-safe نیستند.
- Annotation ها از طرفی بسیار مختصر هستند و قابلیت – compile type – check دارند به همین جهت مدیریت Entity ها را موثر تر میکنند



# Annotations

- @Entity
- @Table
- @Column
- @Id

```
@Entity
@Table(name = "TBL_EMPLOYEE")
public class Employee {
    @Id
    @Column(name="EMPLOYEE_ID")
    private int id =0;
    private String name = null;
    public int getId() {
        return id;
    }
    ...
}
```



# Annotations

- پس از تعریف Annotated کلاس ها آنها را باید در فایل پیکربندی Hibernate نیز تعریف نمود

```
<hibernate-configuration>
    ...
    <mapping class="com.madhusudhan.jh.annotations.Employee"/>
</hibernate-configuration>
```

- همچنین تعریف آنها در کد نیز امکان پذیر است

```
Configuration config = new Configuration()
    .configure("annotations/hibernate.cfg.xml");
    .addAnnotatedClass(Employee.class)
    .addAnnotatedClass(Director.class);
    ...
```



# Annotations

- در تعاریف ستون ها option های بیشتری وجود دارد. به طور مثال اینکه آیا این ستون اختیاری یا اجباری میباشد ، یا اینکه آیا این ستون شرط یکتایی دارد. (unique)


```
@Entity
@Table(name = "TBL_EMPLOYEE")
public class Employee {
    @Id
    @Column(name="EMPLOYEE_ID", nullable = false, unique = true)
    private int employeeId = 0;
    ...
}
```



# Id استراتژی های ساخت

- استراتژی ساخت Id ها به طور پیش فرض AUTO میباشند. در این حالت Hibernate براساس نوع پایگاه داده تان بهترین روش را انتخاب میکند
- برحسب نیازمان میتوانیم استراتژی مذکور را تغییر داد. برای اینکار از @GeneratedValue استفاده میکنیم

```
@Entity
public class Employee {
    @Id
    @Column(name="ID")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int employeeId =0;
    ...
}
```



# انواع استراتژی ها

- GeneratorType.AUTO

- Hibernate بر اساس نوع پایگاه داده انتخاب میکند

- GeneratorType.IDENTITY

- براساس Identity که توسط پایگاه داده تدارک دیده شده است

- GeneratorType.SEQUENCE

- برای پایگاه داده هایی که از Sequence استفاده میکنند

- GeneratorType.TABLE

- از یک جدول برای نگهداری آخرین Id یکتا استفاده میکند



# GeneratorType.SEQUENCE

- در حالتی که بخواهید از استراتژی Sequence استفاده کنید باید از @SequenceGenerator نیز برای تعریف نام Sequence استفاده کنید

```
@Id
@Column(name="EMPLOYEE_ID")
@GeneratedValue (strategy= GenerationType.SEQUENCE, generator="empSeqGen")
@SequenceGenerator(name = "empSeqGen", sequenceName = "EMP_SEQ_GEN")
private int employeeId =0;
```






# GeneratorType.TABLE

- در حالتی که بخواهید از استراتژی Table استفاده کنید باید از @TableGenerator نیز برای تعریف نام جدول مورد نظر استفاده کنید

```
@Id
@Column(name="ID")
@GeneratedValue (strategy= GenerationType.TABLE, generator="empTableGen")
@TableGenerator(name = "empTableGen", table = "EMP_ID_TABLE")
private int employeeId =0;
```



# های ترکیبی Id

- ممکن است همیشه جداول شما از تک ستون ها ( surrogate key ) برای تعریف کلید اصلی استفاده نکنند.
- گهگاه کلید اصلی با ترکیب چند ستون در یک جدول تعریف میشوند ( composite or compound key )
- در این حالت نیاز به مکانیزم دیگری برای تعریف Id در entity ها داریم



# 1 – تعریف کلاس کلید اصلی و @Id

- در این حالت کلاس ثانویه ای تعریف میکنیم که شامل فیلدهای تشکیل دهنده ی کلید اصلی مان باشد.

- @Embeddable
- implements Serializable
- implement hashCode()
- implement equals()

# 1 – تعریف کلاس کلید اصلی و @Id

@Embeddable

```
public class CoursePK implements Serializable{  
    private String tutor = null;  
    private String title = null;  
    // Default constructor  
    public CoursePK() {  
    }  
    ...  
}
```

@Entity

```
@Table(name="COURSE_ANNOTATION")  
public class Course {  
    @Id  
    private CoursePK id = null;
```

## 2 – تعریف کلاس کلید اصلی و @EmbeddedId

- این روش شبیه به روش قبل میباشد و فقط به جای @Id از @EmbeddedId استفاده میکنیم

```
@Entity
@Table(name = "COURSE_ANNOTATION_V2")
public class Course2 {
    @EmbeddedId
    private CoursePK2 id = null;
```



## 3 – استفاده از @IdClass

- در این حالت نیز کلاس ثانویه ساخته میشود با این تفاوت که نیازی به @Embeddable ندارد. سپس @IdClass در بالای سر کلاس entity اضافه شده به فیلدهای متناظر @Id اضافه میکنیم

## 3 – استفاده از @IdClass

```
public class CoursePK3 implements Serializable {  
    private String tutor = null;  
    private String title = null;  
    public CoursePK3() {}  
    ...  
}
```

```
@IdClass(value = CoursePK3.class)  
@Entity  
@Table(name = "COURSE_ANNOTATION_V3")  
public class Course3 {  
  
    // We must duplicate the identifiers  
    // defined in our primary class here too  
    @Id  
    private String title = null;  
    @Id  
    private String tutor = null;  
    ...  
}
```