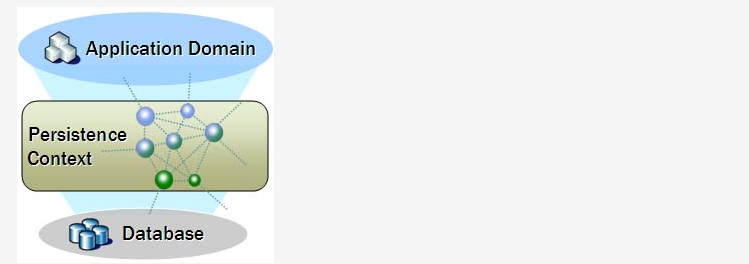
JPA的使用

# JPA概述

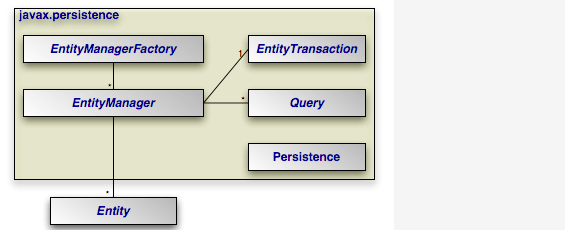
JPA（Java Persistence API，Java持久化API），定义了对象-关系映射（ORM）以及实体对象持久化的标准接口。JPA是JSR-220（EJB3.0）规范的一部分，在JSR-220中规定实体对象（EntityBean）由JPA进行支持。所以JPA不局限于EJB3.0，而是作为POJO持久化的标准规范，可以脱离容器独立运行，开发和测试更加方便。JPA在应用中的位置如下图所示：



JPA维护一个Persistence Context（持久化上下文），在持久化上下文中维护实体的生命周期。主要包含三个方面的内容：

* ORM元数据。JPA支持annotion或xml两种形式描述对象-关系映射。
* 实体操作API。实现对实体对象的CRUD操作。
* 查询语言。约定了面向对象的查询语言JPQL（Java Persistence Query Language）

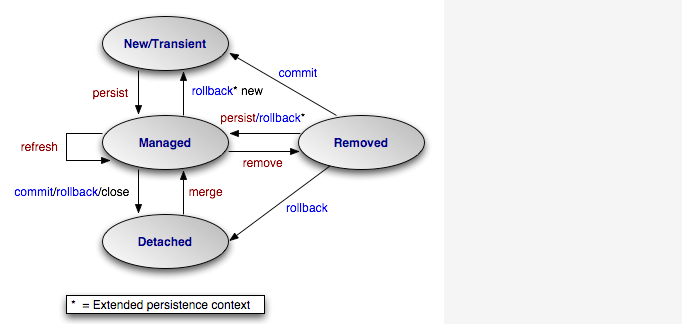
JPA的主要API都定义在javax.persistence包中。如果你熟悉Hibernate，可以很容易做出对应：



| **org.hibernate** | **javax.persistence** | **说明** |
| --- | --- | --- |
| cfg.Configuration | Persistence | 读取配置信息 |
| SessionFactory | EntityManagerFactory | 用于创建会话/实体管理器的工厂类 |
| Session | EntityManager | 提供实体操作API，管理事务，创建查询 |
| Transaction | EntityTransaction | 管理事务 |
| Query | Query | 执行查询 |

# 实体生命周期

实体生命周期是JPA中非常重要的概念，描述了实体对象从创建到受控、从删除到游离的状态变换。对实体的操作主要就是改变实体的状态。JPA中实体的生命周期如下图：



对象状态种类：

* New：新创建的实体对象，没有主键(identity)值
* Managed：对象处于Persistence Context(持久化上下文）中，被EntityManager管理
* Detached：对象已经游离到Persistence Context之外，进入Application Domain
* Removed, 实体对象被删除

EntityManager提供一系列的方法管理实体对象的生命周期，包括：

* persist, 将新创建的或已删除的实体转变为Managed状态，数据存入数据库。
* remove，删除受控实体
* merge，将游离实体转变为Managed状态，数据存入数据库。
* 如果使用了事务管理，则事务的commit/rollback也会改变实体的状态。

# 实体关系映射（ORM）

## 基本映射

| **对象端** | **数据库端** | **annotion** | **可选annotion** |
| --- | --- | --- | --- |
| Class | Table | @Entity | @Table(name="tablename") |
| property | column | – | @Column(name = "columnname") |
| property | primary key | @Id | @GeneratedValue 详见ID生成策略 |
| property | NONE | @Transient |  |

## ID生成策略

ID对应数据库表的主键，是保证唯一性的重要属性。JPA提供了以下几种ID生成策略：

* GeneratorType.AUTO ，由JPA自动生成
* GenerationType.IDENTITY，使用数据库的自增长字段，需要数据库的支持（如SQL Server、MySQL、DB2、Derby等）
* GenerationType.SEQUENCE，使用数据库的序列号，需要数据库的支持（如Oracle）
* GenerationType.TABLE，使用指定的数据库表记录ID的增长 需要定义一个TableGenerator，在@GeneratedValue中引用。例如：@TableGenerator( name="myGenerator", table="GENERATORTABLE", pkColumnName = "ENTITYNAME", pkColumnValue="MyEntity", valueColumnName = "PKVALUE", allocationSize=1 )

@GeneratedValue(strategy = GenerationType.TABLE,generator="myGenerator")

## 关联关系

JPA定义了one-to-one、one-to-many、many-to-one、many-to-many 4种关系。对于数据库来说，通常在一个表中记录对另一个表的外键关联；对应到实体对象，持有关联数据的一方称为owning-side，另一方称为inverse-side。为了编程的方便，我们经常会希望在inverse-side也能引用到owning-side的对象，此时就构建了双向关联关系。 在双向关联中，需要在inverse-side定义mappedBy属性，以指明在owning-side是哪一个属性持有的关联数据。对关联关系映射的要点如下：

| **关系类型** | **Owning-Side** | **Inverse-Side** |
| --- | --- | --- |
| one-to-one | @OneToOne | @OneToOne(mappedBy="othersideName") |
| one-to-many / many-to-one | @ManyToOne | @OneToMany(mappedBy="xxx") |
| many-to-many | @ManyToMany | @ManyToMany(mappedBy ="xxx") |

其中 many-to-many关系的owning-side可以使用@JoinTable声明自定义关联表，比如Book和Author之间的关联表：

@JoinTable(name = "BOOKAUTHOR", joinColumns = { @JoinColumn(name = "BOOKID", referencedColumnName = "id") }, inverseJoinColumns = { @JoinColumn(name = "AUTHORID", referencedColumnName = "id") })

关联关系还可以定制延迟加载和级联操作的行为（owning-side和inverse-side可以分别设置）：

通过设置fetch=FetchType.LAZY 或 fetch=FetchType.EAGER来决定关联对象是延迟加载或立即加载。

通过设置cascade={options}可以设置级联操作的行为，其中options可以是以下组合：

* CascadeType.MERGE 级联更新
* CascadeType.PERSIST 级联保存
* CascadeType.REFRESH 级联刷新
* CascadeType.REMOVE 级联删除
* CascadeType.ALL 级联上述4种操作

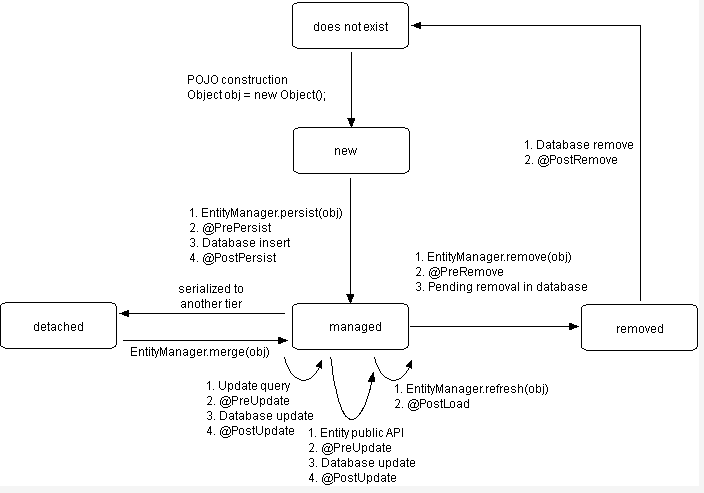
## 继承关系

JPA通过在父类增加@Inheritance(strategy=InheritanceType.xxx)来声明继承关系。A支持3种继承策略：

* 单表继承（InheritanceType.SINGLETABLE），所有继承树上的类共用一张表，在父类指定（@DiscriminatorColumn）声明并在每个类指定@DiscriminatorValue来区分类型。
* 类表继承（InheritanceType.JOINED），父子类共同的部分公用一张表，其余部分保存到各自的表，通过join进行关联。
* 具体表继承（InheritanceType.TABLEPERCLASS)，每个具体类映射到自己的表。

其中1和2能够支持多态，但是1需要允许字段为NULL，2需要多个JOIN关系；3最适合关系数据库，对多态支持不好。具体应用时根据需要取舍。

# 事件及监听



通过在实体的方法上标注@PrePersist，@PostPersist等声明即可在事件发生时触发这些方法。

# Query Language 查询语言

JPA提供两种查询方式，一种是根据主键查询，使用EntityManager的find方法：T find(Class entityClass, Object primaryKey)；另一种就是使用JPQL查询语言。JPQL是完全面向对象的，具备继承、多态和关联等特性，和hibernate HQL很相似。使用EntityManager的createQuery方法：

Query createQuery(String sqlString)

## 使用参数

|  |
| --- |
| #命名参数  Query query = em.createQuery("select p from Person p where p.personid=:Id");  query.setParameter("Id",new Integer(1));  #位置参数  Query query = em.createQuery("select p from Person p where p.personid=?1");  query.setParameter(1,new Integer(1)); |

## 命名查询

如果某个JPQL语句需要在多个地方使用，还可以使用@NamedQuery 或者 @NamedQueries在实体对象上预定义命名查询。在需要调用的地方只要引用该查询的名字即可。

@NamedQuery(name="getPerson", query= "FROM Person WHERE personid=?1")

@NamedQueries({ @NamedQuery(name="getPerson1", query= "FROM Person WHERE personid=?1"), @NamedQuery(name="getPersonList", query= "FROM Person WHERE age>?1") })

Query query = em.createNamedQuery("getPerson");

## 排序

Query query = em.createQuery("select p from Person p order by p.age, p.birthday desc");

## 聚合查询

Query query = em.createQuery("select max(p.age) from Person p"); Object result = query.getSingleResult();

String maxAge = result.toString();

## 更新和删除

JPQL不仅用于查询，还可以用于批量更新和删除。

|  |
| --- |
| Query query = em.createQuery("update Order as o set o.amount=o.amount+10"); //update 的记录数 int result = query.executeUpdate();  Query query = em.createQuery("delete from OrderItem item where item.order in(from Order as o where o.amount<100)"); query.executeUpdate();  query = em.createQuery("delete from Order as o where o.amount<100"); query.executeUpdate();//delete的记录数 |

注意：原生sql语句方式：

@PersistenceContext

private EntityManager entityManager;

entityManager.createNativeQuery(Strign sql);

|  |
| --- |
| @PersistenceContext  private EntityManager entityManager;  StringBuilder sqlDataBuilder = **new** StringBuilder();  Map<String, Object> map = **new** HashMap<String, Object>();  sqlDataBuilder  .append("SELECT c.NAME,a.ORDER\_ID,a.SOURCE,CONCAT(a.SERVE\_DATE,' ',a.BEGIN\_TIME),CONCAT(a.SERVE\_DATE,' ',a.END\_TIME),"  + "b.PAY\_DATE, b.TOTAL\_AMOUNT/100.0,b.DISCOUNT\_AMOUNT/100.0,b.COUPON\_AMOUNT/100.0,"  + "b.CASH\_AMOUNT/100.0,b.CASHCOUPON\_AMOUNT/100.0, "  + "IF(b.THIRD\_PARTY\_PAY\_TYPE=1,b.THIRD\_PARTY\_PAYMENT\_AMOUNT/100.0, NULL), "  + "IF(b.THIRD\_PARTY\_PAY\_TYPE=2,b.THIRD\_PARTY\_PAYMENT\_AMOUNT/100.0, NULL), "  + "b.ACCT\_AMOUNT/100.0, IF(b.THIRD\_PARTY\_PAY\_TYPE=3,b.THIRD\_PARTY\_PAYMENT\_AMOUNT/100.0, NULL), "  + "b.PAY\_STATUS,b.PAY\_REMARK,b.ALIPAY\_TURN/100.0,b.CASH\_TURN/100.0,"  + "b.CASHCOUPON\_TURN/100.0,b.HIT\_ID,b.HIT\_ID\_TYPE" + " FROM ORDER\_MAIN a "  + "JOIN ORDER\_PAYMENT b ON a.ORDER\_ID=b.ORDER\_ID " + "JOIN SERVICE\_INFO c ON a.SERVICE\_ID=c.ID "  + "LEFT JOIN SERVICE\_INFO d ON c.PARENT\_ID=d.ID "  + "LEFT JOIN SERVICE\_INFO e ON d.PARENT\_ID=e.ID "  + "WHERE a.DEL\_FLAG=1 AND a.ORDER\_STATUS='4' ");  StringBuffer conditionBuffer = **new** StringBuffer(); // 公用条件condition  **if** (!BasicUtil.*isEmpty*(dto.getOrderId())) {  conditionBuffer.append(" AND a.ORDER\_ID = :orderId ");  map.put("orderId", dto.getOrderId());  }  Query query = entityManager.createNativeQuery(sqlDataBuilder.append(conditionBuffer).toString());  QueryUtil.*processParamForQuery*(query, map);  List<Object[]> list = query.getResultList();  **return** buildEmployeeIncomeExportDtos(list);  }  **private** List<EmployeeIncomeExportDto> buildEmployeeIncomeExportDtos(List<Object[]> content) {  List<EmployeeIncomeExportDto> list = **new** ArrayList<EmployeeIncomeExportDto>();  **for** (Object[] arr : content) {  EmployeeIncomeExportDto dto = **new** EmployeeIncomeExportDto();  dto.setServiceName(arr[0] == **null** ? **null** : arr[0].toString());  dto.setOrderId(arr[1] == **null** ? **null** : arr[1].toString());  dto.setOrderSource(arr[2] == **null** ? **null** : OrderSource.*getValue*(arr[2].toString()));  dto.setStartTime(arr[3] == **null** ? **null** :  list.add(dto);  } |

# 事务管理

JPA支持本地事务管理（RESOURCELOCAL）和容器事务管理（JTA），容器事务管理只能用在EJB/Web容器环境中。事务管理的类型可以在persistence.xml文件中的“transaction-type”元素配置。JPA中通过EntityManager的getTransaction()方法获取事务的实例（EntityTransaction），之后可以调用事务的begin()、commit()、rollback()方法。