参考：<http://www.jianshu.com/p/afd588c8951c>

**对象持久化API之JPA入门教程**

JPA：Java Persistence API，用于对象持久化的 API

注意：JPA是规范，不是ORM框架，是ORM框架的规范，JPA没有实现ORM，具体实现由ORM厂商提供

现在JPA具体实现框架有：Hibernate、OpenJPA、TopLink(即eclipselink)

**JPA的优势：**

标准化: 提供相同的 API，这保证了基于JPA 开发的企业应用能够经过少量的修改就能够在不同的 JPA 框架下运行。  
简单易用，集成方便: JPA 的主要目标之一就是提供更加简单的编程模型，在 JPA 框架下创建实体和创建 Java 类一样简单，只需要使用 javax.persistence.Entity 进行注释；JPA 的框架和接口也都非常简单，  
可媲美JDBC的查询能力: JPA的查询语言是面向对象的，JPA定义了独特的JPQL，而且能够支持批量更新和修改、JOIN、GROUP BY、HAVING 等通常只有 SQL 才能够提供的高级查询特性，甚至还能够支持子查询。  
支持面向对象的高级特性: JPA 中能够支持面向对象的高级特性，如类之间的继承、多态和类之间的复杂关系，最大限度的使用面向对象的模型

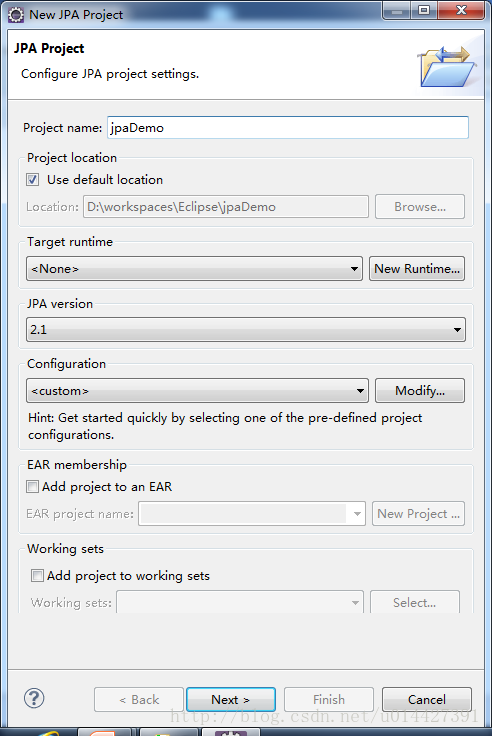
**JPA主要包括这三方面的技术：**

ORM 映射元数据：JPA 支持XML和 JDK 5.0 注解两种元数据的形式，元数据描述对象和表之间的映射关系，框架据此将实体对象持久化到数据库表中。  
JPA 的 API：用来操作实体对象，执行CRUD操作，框架在后台完成所有的事情，开发者从繁琐的 JDBC和 SQL代码中解脱出来。  
查询语言（JPQL）：这是持久化操作中很重要的一个方面，通过面向对象而非面向数据库的查询语言查询数据，避免程序和具体的 SQL 紧密耦合。

**使用JPA实现持久化对象的步骤**

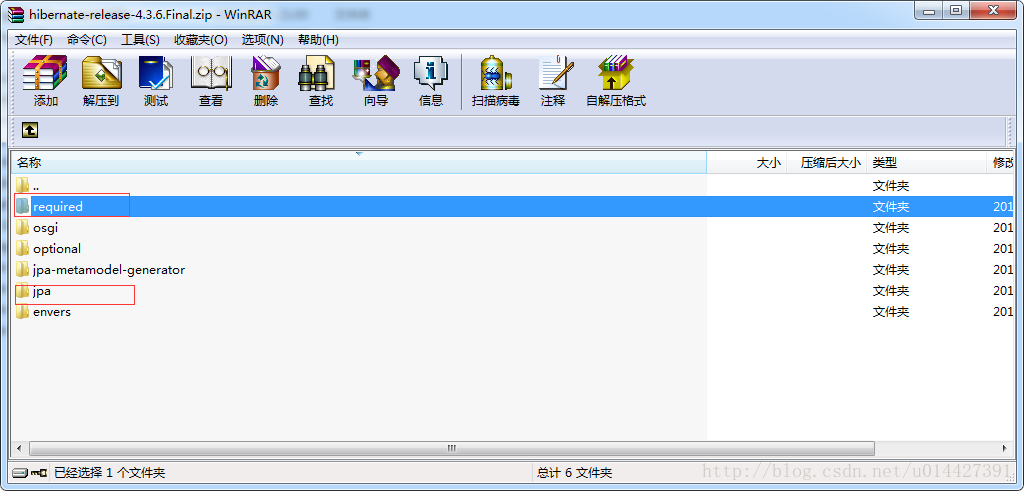
(1)创建 persistence.xml, 在这个文件中配置持久化单元,需要指定跟哪个数据库进行交互;需要指定 JPA 使用哪个持久化的框架以及配置该框架的基本属性  
(2)创建实体类, 使用 annotation 来描述实体类跟数据库表之间的映射关系.  
(3)使用 JPA API 完成数据增加、删除、修改和查询操作  
创建 EntityManagerFactory (对应 Hibernate 中的 SessionFactory);  
创建 EntityManager (对应 Hibernate 中的Session);

使用Eclipse创建一个jpa项目



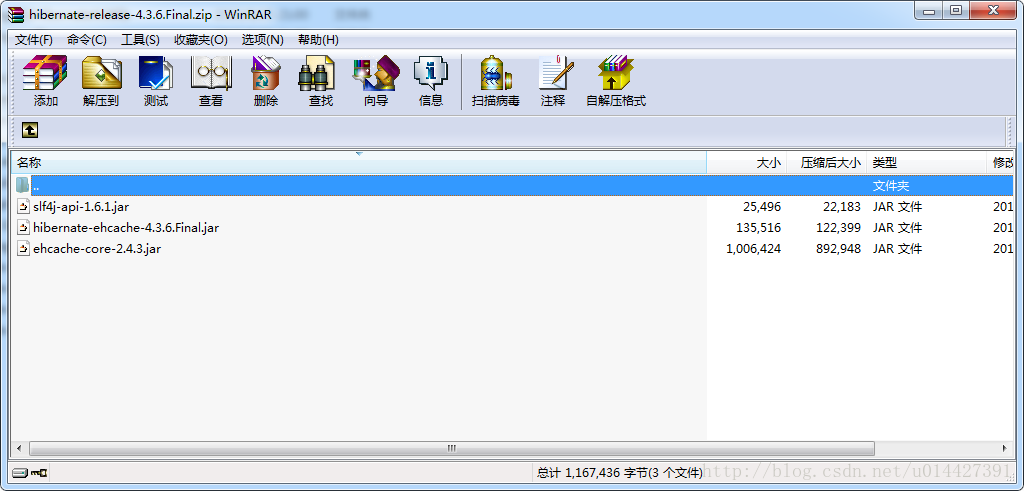
这里写图片描述

加入需要的jar  
hibernate-release-4.3.6.Final\lib\required\*.jar  
hibernate-release-4.3.6.Final\lib\jpa\*.jar



这里写图片描述

echache需要的jar



这里写图片描述

当然还有数据库连接的jar

配置好放在META-INF下面的persistence.xml

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

<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence\_2\_1.xsd">

<persistence-unit name="jpaDemo" transaction-type="RESOURCE\_LOCAL">

<!-- 配置Hibernate JPA作为JPA的实现ORM框架，若项目只有一个JPA实现产品，那么这个配置可以不写 -->

<provider>org.hibernate.ejb.HibernatePersistence</provider>

<!-- 添加持久化类 -->

<class>com.jpaDemo.entity.Customer</class>

<class>com.jpaDemo.entity.Order</class>

<!--

配置二级缓存的策略

ALL：所有的实体类都被缓存

NONE：所有的实体类都不被缓存.

ENABLE\_SELECTIVE：标识 @Cacheable(true) 注解的实体类将被缓存

DISABLE\_SELECTIVE：缓存除标识 @Cacheable(false) 以外的所有实体类

UNSPECIFIED：默认值，JPA 产品默认值将被使用

-->

<shared-cache-mode>ENABLE\_SELECTIVE</shared-cache-mode>

<properties>

<!-- 连接数据库的基本信息 -->

<property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>

<property name="javax.persistence.jdbc.url" value="jdbc:mysql:///jpa"/>

<property name="javax.persistence.jdbc.user" value="root"/>

<property name="javax.persistence.jdbc.password" value="root"/>

<!-- 配置 JPA 实现产品的基本属性. 配置 hibernate 的基本属性 -->

<property name="hibernate.format\_sql" value="true"/>

<property name="hibernate.show\_sql" value="true"/>

<property name="hibernate.hbm2ddl.auto" value="update"/>

<!-- 二级缓存相关 -->

<property name="hibernate.cache.use\_second\_level\_cache" value="true"/>

<property name="hibernate.cache.region.factory\_class" value="org.hibernate.cache.ehcache.EhCacheRegionFactory"/>

<property name="hibernate.cache.use\_query\_cache" value="true"/>

</properties>

</persistence-unit>

</persistence>

echache.xml:

<ehcache>

<!-- Sets the path to the directory where cache .data files are created.

If the path is a Java System Property it is replaced by

its value in the running VM.

The following properties are translated:

user.home - User's home directory

user.dir - User's current working directory

java.io.tmpdir - Default temp file path -->

<diskStore path="java.io.tmpdir"/>

<!--Default Cache configuration. These will applied to caches programmatically created through

the CacheManager.

The following attributes are required for defaultCache:

maxInMemory - Sets the maximum number of objects that will be created in memory

eternal - Sets whether elements are eternal. If eternal, timeouts are ignored and the element

is never expired.

timeToIdleSeconds - Sets the time to idle for an element before it expires. Is only used

if the element is not eternal. Idle time is now - last accessed time

timeToLiveSeconds - Sets the time to live for an element before it expires. Is only used

if the element is not eternal. TTL is now - creation time

overflowToDisk - Sets whether elements can overflow to disk when the in-memory cache

has reached the maxInMemory limit.

-->

<defaultCache

maxElementsInMemory="10000"

eternal="false"

timeToIdleSeconds="120"

timeToLiveSeconds="120"

overflowToDisk="true"

/>

<!--Predefined caches. Add your cache configuration settings here.

If you do not have a configuration for your cache a WARNING will be issued when the

CacheManager starts

The following attributes are required for defaultCache:

name - Sets the name of the cache. This is used to identify the cache. It must be unique.

maxInMemory - Sets the maximum number of objects that will be created in memory

eternal - Sets whether elements are eternal. If eternal, timeouts are ignored and the element

is never expired.

timeToIdleSeconds - Sets the time to idle for an element before it expires. Is only used

if the element is not eternal. Idle time is now - last accessed time

timeToLiveSeconds - Sets the time to live for an element before it expires. Is only used

if the element is not eternal. TTL is now - creation time

overflowToDisk - Sets whether elements can overflow to disk when the in-memory cache

has reached the maxInMemory limit.

-->

<!-- Sample cache named sampleCache1

This cache contains a maximum in memory of 10000 elements, and will expire

an element if it is idle for more than 5 minutes and lives for more than

10 minutes.

If there are more than 10000 elements it will overflow to the

disk cache, which in this configuration will go to wherever java.io.tmp is

defined on your system. On a standard Linux system this will be /tmp"

-->

<cache name="sampleCache1"

maxElementsInMemory="10000"

eternal="false"

timeToIdleSeconds="300"

timeToLiveSeconds="600"

overflowToDisk="true"

/>

<!-- Sample cache named sampleCache2

This cache contains 1000 elements. Elements will always be held in memory.

They are not expired. -->

<cache name="sampleCache2"

maxElementsInMemory="1000"

eternal="true"

timeToIdleSeconds="0"

timeToLiveSeconds="0"

overflowToDisk="false"

/> -->

<!-- Place configuration for your caches following -->

</ehcache>

实体类：

package com.jpaDemo.entity;

import java.util.Date;

import java.util.HashSet;

import java.util.Set;

import javax.persistence.Cacheable;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.FetchType;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.NamedQuery;

import javax.persistence.OneToMany;

import javax.persistence.Table;

import javax.persistence.Temporal;

import javax.persistence.TemporalType;

import javax.persistence.Transient;

@Cacheable(true)

@Table(name="JPA\_CUTOMERS")

@Entity

publicclassCustomer {

private Integer id;

private String lastName;

private String email;

privateint age;

private Date createdTime;

private Date birth;

publicCustomer() {

// TODO Auto-generated constructor stub

}

publicCustomer(String lastName, int age) {

super();

this.lastName = lastName;

this.age = age;

}

private Set<Order> orders = new HashSet<>();

// @TableGenerator(name="ID\_GENERATOR",

// table="jpa\_id\_generators",

// pkColumnName="PK\_NAME",

// pkColumnValue="CUSTOMER\_ID",

// valueColumnName="PK\_VALUE",

// allocationSize=100)

// @GeneratedValue(strategy=GenerationType.TABLE,generator="ID\_GENERATOR")

@GeneratedValue(strategy=GenerationType.AUTO)

@Id

public Integer getId() {

return id;

}

publicvoidsetId(Integer id) {

this.id = id;

}

@Column(name="LAST\_NAME",length=50,nullable=false)

public String getLastName() {

return lastName;

}

publicvoidsetLastName(String lastName) {

this.lastName = lastName;

}

public String getEmail() {

return email;

}

publicvoidsetEmail(String email) {

this.email = email;

}

publicintgetAge() {

return age;

}

publicvoidsetAge(int age) {

this.age = age;

}

@Temporal(TemporalType.TIMESTAMP)

public Date getCreatedTime() {

return createdTime;

}

publicvoidsetCreatedTime(Date createdTime) {

this.createdTime = createdTime;

}

@Temporal(TemporalType.DATE)

public Date getBirth() {

return birth;

}

publicvoidsetBirth(Date birth) {

this.birth = birth;

}

//映射单向 1-n 的关联关系

//使用 @OneToMany 来映射 1-n 的关联关系

//使用 @JoinColumn 来映射外键列的名称

//可以使用 @OneToMany 的 fetch 属性来修改默认的加载策略

//可以通过 @OneToMany 的 cascade 属性来修改默认的删除策略.

//注意: 若在 1 的一端的 @OneToMany 中使用 mappedBy 属性, 则 @OneToMany 端就不能再使用 @JoinColumn 属性了.

// @JoinColumn(name="CUSTOMER\_ID")

@OneToMany(fetch=FetchType.LAZY,cascade={CascadeType.REMOVE},mappedBy="customer")

public Set<Order>getOrders() {

return orders;

}

publicvoidsetOrders(Set<Order> orders) {

this.orders = orders;

}

//工具方法. 不需要映射为数据表的一列.

@Transient

public String getInfo(){

return"lastName: " + lastName + ", email: " + email;

}

@Override

public String toString() {

return"Customer [id=" + id + ", lastName=" + lastName + ", email="

+ email + ", age=" + age + ", createdTime=" + createdTime

+ ", birth=" + birth + "]";

}

}

//================20170731

测试类：

package com.jpaDemo.test;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.EntityTransaction;

import javax.persistence.Persistence;

import com.jpaDemo.entity.Customer;

publicclassMyTest {

publicstatic void main(String[] args) {

//1. 创建 EntitymanagerFactory

String persistenceUnitName = "jpaDemo";

Map<String, Object> properites = newHashMap<String, Object>();

properites.put("hibernate.show\_sql", true);

EntityManagerFactory entityManagerFactory =

Persistence.createEntityManagerFactory(persistenceUnitName, properites);

//2. 创建 EntityManager. 类似于 Hibernate 的 SessionFactory

EntityManager entityManager = entityManagerFactory.createEntityManager();

//3. 开启事务

EntityTransaction transaction = entityManager.getTransaction();

transaction.begin();

//4. 进行持久化操作

Customer customer = newCustomer();

customer.setAge(12);

customer.setEmail("123@qq.com");

customer.setLastName("Tom");

customer.setBirth(newDate());

customer.setCreatedTime(newDate());

entityManager.persist(customer);

//5. 提交事务

transaction.commit();

//6. 关闭 EntityManager

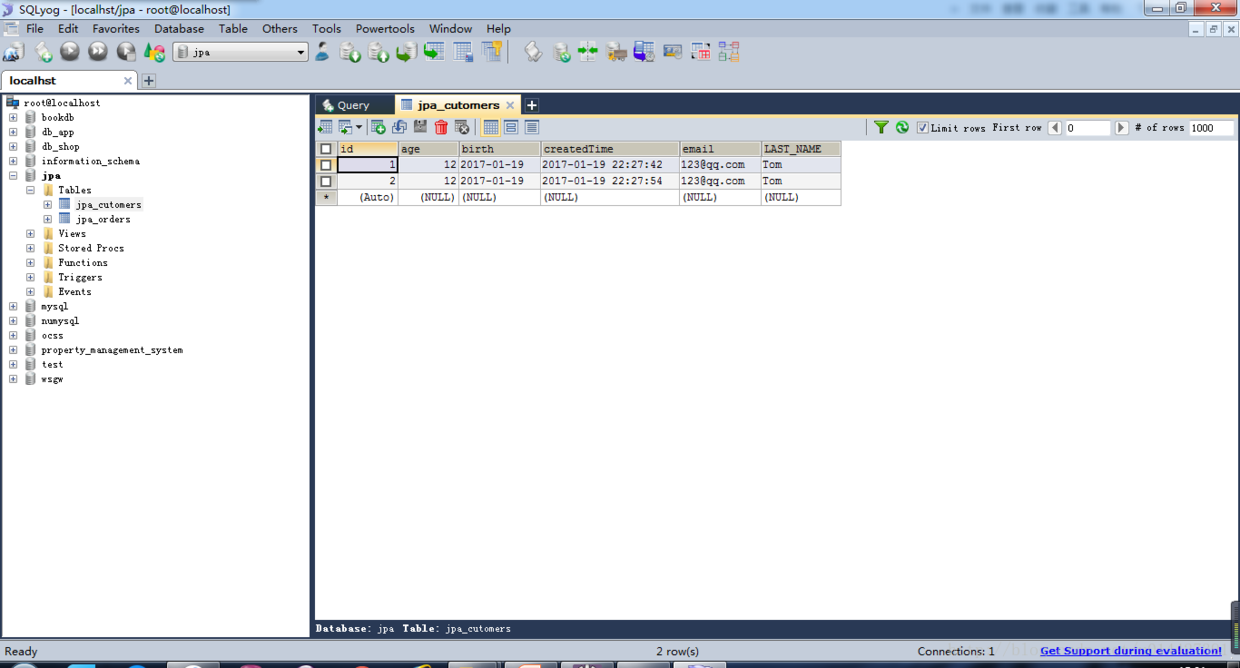
entityManager.close();

//7. 关闭 EntityManagerFactory

entityManagerFactory.close();

}

}



这里写图片描述