Atitit jpa的使用总结

目录

[1. JPA包含的技术 1](#_Toc3215)

[1.1. JPA的供应商 1](#_Toc5760)

[1.2. Pom maven 1](#_Toc26223)

[1.3. JpaOpenjpa 2](#_Toc22992)

[1.4. JPA的persistence.xml文件 3](#_Toc4304)

[1.5. Other 5](#_Toc18518)

[1.6. Custome jpa cfg 7](#_Toc7264)

[2. Ref 8](#_Toc16347)

# JPA包含的技术

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。

————————————————

## JPA的供应商

JPA 的目标之一是制定一个可以由很多供应商实现的 API，Hibernate 3.2+、TopLink 10.1+ 以及 OpenJPA 都提供了 JPA 的实现，JPA 供应商有很多，常见的有如下四种：

1.Hibernate

JPA 的始作俑者就是 Hibernate 的作者，Hibernate 从 3.2 开始兼容 JPA。

2.OpenJPA

OpenJPA 是 Apache 组织提供的开源项目。

3.TopLink

TopLink 以前需要收费，如今开源了。

4.EclipseLink

————————————————

## Pom maven

<!-- https://mvnrepository.com/artifact/org.apache.openjpa/openjpa -->

<dependency>

<groupId>org.apache.openjpa</groupId>

<artifactId>openjpa</artifactId>

<version>3.1.1</version>

</dependency>

<!-- https://mvnrepository.com/artifact/javax.persistence/javax.persistence-api -->

<dependency>

<groupId>javax.persistence</groupId>

<artifactId>javax.persistence-api</artifactId>

<version>2.2</version>

</dependency>

## JpaOpenjpa

**import** java.util.List;

**import** java.util.Map;

**import** javax.persistence.EntityManager;

**import** javax.persistence.EntityManagerFactory;

**import** javax.persistence.Persistence;

**import** javax.persistence.Query;

**import** com.alibaba.fastjson.JSON;

**public** **class** JpaOpenjpa {

**public** **static** **void** main(String[] args) {

// Create a new EntityManagerFactory using the System properties.

// The "hellojpa" name will be used to configure based on the

// corresponding name in the META-INF/persistence.xml file

EntityManagerFactory factory = Persistence.*createEntityManagerFactory*("wmsPersisteUnitName");

// Create a new EntityManager from the EntityManagerFactory. The

// EntityManager is the main object in the persistence API, and is

// used to create, delete, and query objects, as well as access

// the current transaction

EntityManager em = factory.createEntityManager();

// Perform a simple query for all the Message entities

Query q = em.createNativeQuery("select \* from sys\_tab",Map.**class**);

List li = q.getResultList();

System.***out***.println(JSON.*toJSONString*(li));

}

}

## [JPA的persistence.xml文件](https://www.cnblogs.com/luxh/archive/2012/05/24/2516282.html)

Posted on 2012-05-24 12:27  [CN.programmer.Luxh](https://www.cnblogs.com/luxh/)  阅读(20202)  评论(0)  [编辑](https://i.cnblogs.com/EditPosts.aspx?postid=2516282)  [收藏](https://www.cnblogs.com/luxh/archive/2012/05/24/javascript:void(0))

　　persistence.xml文件必须定义在classpath路径下的META-INF文件夹中。

IMG_256

　　我们看看基于Hibernate提供的一个比较完整的JPA2.0的persistence.xml文件。

　　persistence.xml:

　要注意使用的是2.0规范

**name**

　　JPA2.0规范要求每一个持久化单元必须有一个名字，不能为空。即persistence-unit name="manager1"的name不能为空。

　　transaction-type

　　使用的事务类型。有JTA和RESOURCE\_LOCAL两种类型可以选择。在JavaEE环境中默认为JTA,在JavaSE环境中默认为RESOURCE\_LOCAL。当在persistent.xml文件使用<jta-data-source>,默认就是JTA事务，使用<non-jta-data-source>，默认就是使用RESOURCE\_LOCAL事务。这两种事务的区别不在这里讨论。

　　provider

　　EJB Persistence provider的一个实现类。如果不是使用多个厂商的 EJB Persistence实现，是不需要定义的。

　　mapping-file

　　指定映射文件的位置

　　jar-file

　　指定要解析的jar。jar中所有注解的类、包和所有的hbm.xml都会被添加到persistent-unit的配置中。主要用在JavaEE环境中。

　　exclude-unlisted-classes

　　不检查jar中加了@Entity注解的类。

　　class

　　明确指定要映射的类

　　shared-cache-mode

　　缓存模式。加了@Cacheable注解的默认为二级缓存。有四种模式：ALL-缓存所有实体；NONE-禁止缓存；ENABLE\_SELECTIVE-如果加了缓存的标识，是默认的选选　　　　　　　　项；DISABLE\_SELECTIVE- enable caching unless explicitly marked as  @Cacheable(false) (not  recommended)

　　validation-mode

　　实体的验证模式，默认是激活的。当一个实体在创建、更新，在实体发送到数据库前会被进行验证。CALLBACK: entities are validated on creation, update and deletion. If no Bean Validation provider  is present, an exception is raised at initialization time.

　　properties

　　配置厂商的一些特定属性。

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

<persistence xmlns=*"http://java.sun.com/xml/ns/persistence"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* version=*"2.0"*>

<persistence-unit name=*"wmsPersisteUnitName"*>

<provider>org.apache.openjpa.persistence.PersistenceProviderImpl

</provider>

<!--

<class>com.wms.jpa.model.User</class>

<class>com.wms.jpa.model.Message</class>

openjpa.ConnectionDriverName or javax.persistence.jdbc.driver

-->

<properties>

<property name=*"openjpa.ConnectionDriverName"* value=*"com.mysql.cj.jdbc.Driver"* />

<property name=*"openjpa.ConnectionURL"* value=*"jdbc:mysql llowMultiQueries=true"* />

<property name=*"openjpa.ConnectionUserName"* value=*" "* />

<property name=*"openjpa.ConnectionPassword"* value=*" "* />

</properties>

</persistence-unit>

<!-- <property name="openjpa.Multithreaded" value="true" />

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

<property name="openjpa.DynamicEnhancementAgent" value="false" />

<property name="openjpa.RuntimeUnenhancedClasses" value="unsupported" />

<property name="openjpa.ConnectionFactoryProperties" value="PrintParameters=true" />

<property name="openjpa.jdbc.SynchronizeMappings" value="buildSchema(ForeignKeys=True)" /> -->

</persistence>

## Other

Hb的和openjpa的使用api逗死jpa一样的

Spring data api失败

package com.kok.sport.utils.db;

import java.util.List;

import java.util.Map;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.core.type.AnnotationMetadata;

import org.springframework.data.repository.config.AnnotationRepositoryConfigurationSource;

import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)

@SpringBootTest

public class JpaTest {

@Autowired

JpaDao JpaDao1;

@Test

public void testAdd() {

// AnnotationMetadata

List<Map> li= JpaDao1.findSQL("1", "2");

System.out.println(li);

System.out.println("F");

}

public static void main(String[] args) {

//AnnotationRepositoryConfigurationSource

}

}

package com.kok.sport.utils.db;

import java.util.List;

import java.util.Map;

import java.util.Optional;

import org.springframework.data.domain.Example;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.domain.Sort;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.query.Param;

@Repository

public interface JpaDao extends JpaRepository<Map , Long>{

//@Query(nativeQuery = true, value = "SELECT \* FROM AUTH\_USER WHERE name = :name1 OR name = :name2 ")

@Query(nativeQuery = true, value = "SELECT 333 ")

List<Map> findSQL(@Param("name1") String name1, @Param("name2") String name2);

}

## Custome jpa cfg

**private** **static** EntityManagerFactory JpaUtil\_getFac() {

String url2;// =

url2=MybatisUtil. *getMysqlUrl*();

System.***out***.println(url2);

String mysqlConnUrl =url2;

ConnectionUrlParser connStringParser = ConnectionUrlParser.*parseConnectionString*(mysqlConnUrl);

System.***out***.println(connStringParser);

Object url = mysqlConnUrl;

Object usr = UrlUtil.*parse4Q*(connStringParser.getQuery()).get("user");

Object pwd = UrlUtil.*parse4Q*(connStringParser.getQuery()).get("password");

Map properties = Maps.*newLinkedHashMap*();

properties.put("javax.persistence.jdbc.driver", "com.mysql.cj.jdbc.Driver");

properties.put("javax.persistence.jdbc.url",url);

properties.put("javax.persistence.jdbc.user", usr.toString()); // if needed

properties.put("javax.persistence.jdbc.password", pwd.toString()); // if needed

System.***out***.println(properties);

// Create a new EntityManagerFactory using the System properties.

// The "hellojpa" name will be used to configure based on the

// corresponding name in the META-INF/persistence.xml file

EntityManagerFactory factory = Persistence.*createEntityManagerFactory*(**null**,properties);

**return** factory;

}

# Ref

(···条消息)5 Hibernate：Java Persistence API (JPA) 入门\_java\_Silent\_Paladin的博客-CSDN博客