【】Hibernate概念

Hibernate是JavaEE中DAO层框架

Hiberate是ORM框架，解决方案。

【】Hibernate功能

Model对象的持久化操作：CUDR

C: insert

U:update

D: delete

R: Read/Retrive -Select

Model对象<-->DAO<-->表记录

【】ORM原理

ORM-Object Relation Mapping

对象世界--- 关系世界

类---表

对象--记录

属性--字段

完成以上操作的框架称为ORM框架

【】ORM框架种类

1.Hibernate

2.Mybatis

3.JPA

4.TopLink

5.JDO

【】Hibernate组成

1.接口：

(1) SessionFactory

(2) Session

(3) Transaction

(4) Query

2.工厂类：

Configuration

3.配置文件：配置数据库连接信息。

4.映射信息；完成Model类与表的映射配置。

【】Hibernate应用开发步骤

1.创建动态Web项目

2.创建数据库

create database cityoa;

3.创建表

create table OA\_Department

(

DEPTNO int(10) primary key auto\_increment,

DEPTCODDE varchar(20) not null,

DEPTNAME varchar(100)

)

4.创建Model类：

包和类的命名规则参考上学期的JavaEE课程。

package com.city.oa.model;

public class DepartmentModel {

private int no=0;

private String code=null;

private String name=null;

public int getNo() {

return no;

}

public void setNo(int no) {

this.no = no;

}

public String getCode() {

return code;

}

public void setCode(String code) {

this.code = code;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

第2讲 Hibernate配置

【】授课内容

1.Hibernate配置的功能

2.Hibernate配置方式

3.XML配置方式文件命名，存储，语法。

4.属性文件配置方式文件命名，存储和语法。

5.编程配置方式的简单编程。

6.Hibernate常见的配置属性。

7.Hibernate配置数据库连接的方式。

8.取得SessionFactory的方式及编程

【】Hibernate配置的功能

1.配置数据库的连接

2.配置连接池的参数

3.配置使用数据库的类型（方言-Dialect）

4.配置映射文件的位置（XML映射方式）。

5.配置Model的包和类。

6.配置缓存机制。

7.检索机制配置。

【】Hibernate配置方式

1.XML方式

2.属性文件方式

3.编程方式

【】XML方式

1.默认的XML配置文件：hibernate.cfg.xml

读取:cfg.configure();

2.位置：classpath根目录

3.自己定义XML文件；lhd.xml

读取：cfg.configure("lhd.xml");

4. 语法：

<session-factory>

<!-- Database connection settings -->

<property name="connection.driver\_class">com.mysql.jdbc.Driver</property>

<property name="connection.url">jdbc:mysql://localhost:3306/cityoa</property>

<property name="connection.username">root</property>

<property name="connection.password">root</property>

<!-- JDBC connection pool (use the built-in) -->

<property name="connection.pool\_size">1</property>

<!-- SQL dialect -->

<property name="dialect">org.hibernate.dialect.MySQLDialect</property>

<!-- Disable the second-level cache -->

<property name="cache.provider\_class">org.hibernate.cache.internal.NoCacheProvider</property>

<!-- Echo all executed SQL to stdout -->

<property name="show\_sql">true</property>

<mapping resource="com/city/oa/model/DepartmentModel.hbm.xml"/>

<mapping resource="com/city/oa/model/EmployeeModel.hbm.xml"/>

</session-factory>

注：XML方式属性名前缀hibernate.可以省略。

【】属性文件方式

1.默认的属性文件名：hibernate.properties

读取：

Configuration cfg=new Configuration();

2.默认的位置：classpath根目录

3.语法：属性名=值

4.案例：

hibernate.connection.driver\_class=com.mysql.jdbc.Driver

hibernate.connection.url=jdbc:mysql://localhost:3307/cityoa

hibernate.connection.username=root

hibernate.connection.password=root

注：属性文件方式，前缀hibernate.不能省略。

5.优点：

语法简单。

6.缺点：

无法引入映射文件。

【】编程方式

1.使用Configuration对象方法进行。

cfg.setProperty("hibernate.connection.pool\_size", "2");

cfg.addResource("com/city/oa/model/DepartmentModel.hbm.xml");

【】常见的配置属性

1.数据库连接

2.连接池配置

(1)Hibernate自带的连接池配置

<property name="connection.pool\_size">1</property>

1. 使用Tomcat配置的连接池

第03讲 Hibernate简单映射与主键生成器

【】授课内容

1.Hibernate映射类型

2.XML映射基本语法

3.Java注释映射基本语法

4.Hibernate主键生成器类型

5.XML主键生成器配置

6.Java注释生成器配置。

【】Hibernate映射类型

1.XML映射方式

2.Java注释方式

【】XML映射基本语法

1.XML映射的原则

(1)每个Model类有自己的XML映射文件。

(2)命名规范：Model类名.hbm.xml

(3)存储：与Model类同包。

2.语法：

<hibernate-mapping package="com.city.oa.model">

<class name="DepartmentModel" table="OA\_Department">

<id name="no" column="DEPTNO">

<generator class="identity"/>

</id>

<property name="code" column="DEPTCODE"/>

<property name="name" column="DEPTNAME"/>

</class>

</hibernate-mapping>

3.<hibernate-mapping>

(1) package=""

4.<class> 映射类与表对应

(1)name="类名"

(2)table="表名"

(3)dynamic-insert="true|false"

(4) dynamic-update="true|false"

(5) lazy="true|false"

5.<id> 映射主键字段属性的。

(1) name="主属性名"

(2) column="字段名"

(3) type="类型"

type="date"

type="time"

access="field | property | classname"

6.<property>映射简单属性（非PK非FK）。

(1) name="主属性名"

(2) column="字段名"

(3) type="类型"

(4) update="true|false":

(5) insert=“true|false”

(6) lazy=""

(7) unique="true"

(8) not-null="true"

7.<generator>

【】Java注释映射基本语法

1.类的映射注释类

(1) @Entity: 持久类声明

(2) @Table(name="表名") 指定类对应的表名

2.主属性映射

(1) @Id

(2) @Column(name="字段名")

3.普通属性映射

(1) @Basic

(2) @Column(name="字段名")

【】Hibernate主键生成器类型

1.数据库负责生成主键值的类型

(1)identity：由数据库的自增量字段负责。

(2)sequence：数据库的序列值。

(3)guid：由数据库生成GUID字符串。

(4)native：让数据库自行选择合适的生成器。

(5)select：数据库的触发器生成主键值。

2.Hibernate负责的生成主键值的类型

(1) increment

Hibernate执行select max(主键字段)+1 from 表

(2) hilo：

(3) seqhilo：

(4) uuid：运行UUID算法生成符合UUID字符串。

(5) foreign：由HIbernate根据一对一关联设置主键值。

(6) sequence-identity：

3.应用程序负责的生成主键值的类型

(1)assiged:

【】XML主键生成器配置

1.语法：

(1)

<generator class="assigned"/>

(2)

<generator class="sequnce">

<param name="sequnce">SQ\_DEPT</param>

</generator>

【】Java注释生成器配置

1.使用@GeneratedValue

2.语法：

@GeneratedValue(strategy = GenerationType.IDENTITY)

第04讲 关联映射-多对一和一对多

【】授课内容

1.Java对象关系类型，意义，表达

2.关联关系的关系模型表达

3.多对一关联关系的Hibernate映射

(1)XML方式映射

(2)Java注释方式映射

4.一对多关联关系的Hibernate映射

(1)XML方式映射

(2)Java注释方式映射

5.多对一和一对多关联关系的应用编程

【】Java对象关系类型，意义，表达

1.依赖关系(Dependency)

(1) 表达意义：使用，（use。。。）

(2) Java表达：方法中使用另一个类的对象

class A{

public void a(){}

}

class B {

public void b(){

A a=new A();

a.a();

}

public void b(A a){

a.a();

}

public A createA(){

return new A();

}

}

2.继承关联(Inhirent)

(1)意义：是的关系(is ....)

(2) Java表达

class A extends B {

}

3.实现关系(implements)

(1)意义：遵守的关系( ....)

(2) Java表达:

class DepartmentServiceImpl implements IDepartmentService

{

}

4.关联关系（Assoiation）

(1)意义：拥有的关系(have /has ....)

(2) Java表达:

class A{

public void a(){}

}

class B {

private A a=null;

}

(3)关系的特性：

<1>数量性：一个，多个。

<2>方向性：单向，双向。

(4)案例：

<1>B有3个A，A有一个B

class A{

private B b=null;

}

class B {

private A[] as=new A[3];

}

<2>B有多个A，A有一个B

class A{

private B b=null;

}

class B {

private Set<A> as=new HashSet<A>();

}

<3>B有多个A，A有多个个B

class A{

private Set<B> bs=new HashSet<B>();

}

class B {

private Set<A> as=new HashSet<A>();

}

【】多对一和一对多的关系模型表达

1.Java表达

class A{

private B b=null;

}

class B {

private Set<A> as=new HashSet<A>();

}

2.数据库表达：

多方表有一个Fk指向一方表的PK。

【】多对一关联关系的Hibernate映射

0.Java定义

public class EmployeeModel {

private String id=null;

private String password=null;

private String name=null;

private String sex=null;

private int age=0;

private Date birthday=null;

private Date joinDate=null;

private double salary=0;

//关联的多对一部门

private DepartmentModel department=null;

}

1. XML映射：

<!-- 表达员工对部门的多对一关联关系 -->

<many-to-one name="department" column="DEPTNO" class="DepartmentModel" />

2. Java注释映射

【】一对多关联关系的Hibernate映射

0. Java表达

public class DepartmentModel {

//部门编号

@Id

@Column(name="DEPTNO")

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int no=0;

//部门编码

@Basic

@Column(name="DEPTCODE")

private String code=null;

//部门名称

@Column(name="DEPTNAME")

private String name=null;

//一对多关联属性

private Set<EmployeeModel> employees=null;

}

1. XML映射

<set name="employees" table="OA\_Employee">

<key column="DEPTNO" />

<one-to-many class="EmployeeModel" />

</set>

1. Java注释映射

第05讲 多对多和一对一映射

【】授课大纲

1.多对多关联的Java表达，数据库表达

2.多对多关联的Hibernate映射

(1)XML方式映射语法

(2)Java注释方式映射语法

3.多对多关联的实际应用编程

4.一对一关联关系的Java表达，数据库表达

5.一对一关联关系的Hibernate映射

(1)XML方式映射语法

(2)Java注释方式映射语法

6.一对一关联关系的实际编程应用

【】多对多关联的Java表达，数据库表达

1.多对多关联的Java表达

class A {

private Set<B> as=new HashSet<B>();

}

class B {

private Set<A> as=new HashSet<A>();

}

2多对多关联的数据库表达

使用关联表。

只有2个字段，分别来自关联表的主键。

这2个字段都是FK，指向关联的PK。

这2个字段联合做主键。

【】多对多关联的Hibernate映射

1. XML方式

<set name="behaves" table="oa\_employeebehave" >

<key column="EMPID" />

<many-to-many column="BNO" class="BehaveModel" />

</set>

<set name="employees" table="oa\_employeebehave">

<key column="BNO" />

<many-to-many column="EMPID" class="EmployeeModel" />

</set>

2.Java注释方式

//

@ManyToMany

@JoinTable(

name="oa\_employeebehave",

joinColumns={@JoinColumn (name="EMPID")},

inverseJoinColumns={@JoinColumn(name="BNO")}

)

private Set<BehaveModel> behaves=null;

//

@ManyToMany(mappedBy="behaves")

private Set<EmployeeModel> employees=null;

【】多对多关联的实际应用编程

1.为关联表增加数据

案例：为指定的员工增加爱好，

或为指定的爱好增加员工

//为指定的员工增加爱好

public void addBehave(String id, int behaveNo) throws Exception {

SessionFactory sf=HibernateFactory.getSessionFactory();

Session session=sf.openSession();

Transaction tx=session.beginTransaction();

EmployeeModel em=session.get(EmployeeModel.class, id);

BehaveModel bm=session.get(BehaveModel.class, behaveNo);

em.getBehaves().add(bm);

//bm.getEmployees().add(em);

tx.commit();

session.close();

}

//一次增加多个爱好

public void addBehaves(String id, int[] behaveNos) throws Exception {

SessionFactory sf=HibernateFactory.getSessionFactory();

Session session=sf.openSession();

Transaction tx=session.beginTransaction();

EmployeeModel em=session.get(EmployeeModel.class, id);

for(int bno:behaveNos){

BehaveModel bm=session.get(BehaveModel.class, bno);

em.getBehaves().add(bm);

//bm.getEmployees().add(em);

}

tx.commit();

session.close();

}

2.取得指定对象关联的多方集合

案例：取得指定员工的爱好列表，

或取得指定爱好的员工列表

【】一对一关联关系的Java表达，数据库表达

1.Java表达：

class A {

private B b=null;

}

class B {

private A a=null;

}

2.数据库表达

(1)PK做FK。

(2) 专门FK指向PK，FK要求唯一性。

【】一对一关联关系的Hibernate映射

(1)XML方式映射语法

(2)Java注释方式映射语法

第06讲 Hibernate持久化编程

【】授课大纲

1.Hibernate的持久化接口Session及主要方法

2.Hibernate持久对象的状态和转换方法

3.Hibernate持久对象的检索策略对持久化的影响

【】Hibernate的持久化接口Session及主要方法类型

Session功能执行CUDR

1.操作持久对象：CUDR。

2.管理连接

3.管理缓存。

4.管理查询对象Query。

【】Hibernate的持久化接口Session的持久对象方法

1.增加(C-Create, 生成insert into )

(1) save(Model对象)

(2) saveOrUpdate(Model对象)

(3) persist(Model对象)

附加：save与persist区别

<1> save能返回对象的主键值，persist无返回。

<2> save把增加的持久对象保存到session缓存，

persist不保存对象到session缓存。

<3> save能操作任何状态的持久对象，

persist只能操作临时态和持久态。

2.修改(U-Update 生成update语句）

(1) update(Model对象)

(2) saveOrUpdate(Model对象)

(3) merge(Model对象)

3.删除(D-Delete 生成delete from 语句）

(1)delete(Model对象)

4.查询(R-Retrive或Read 生成select语句)

Session的查询方法只能根据主键值取得一个持久对象。

(1) Model对象 session.get(Model.class,主键值)

(2) Model对象 session.load(Model.class,主键值)

附加:get和load区别

<1> 如果主键值不存在，get返回null, load抛异常。

<2> 如果持久对象映射时设定class为延迟检索，

load不执行select, 返回Model的代理对象，

只有主属性有值。

get不管lazy属性，执行立即检索，返回Model对象实例。

<3> load方法先查询一级和二级缓存，

get只查询一级缓存。

【】Hibernate持久对象的状态和转换方法

1.临时态

(1) JVM内存中存在。

(2) 表无对应记录。

(3) 没有关联session. （没有在Hibernate的管理状态）。

2.持久态

(1) JVM内存中存在。

(2) 表有对应记录。

(3) 关联session. （在Hibernate的管理状态）。

3.游离态

(1) JVM内存中存在。

(2) 表有对应记录。

(3) 无关联的session. （没有在Hibernate的管理状态）。

第07讲 Hibernate简单查询

【】授课内容

1.Hibernate查询机制类型

2.HQL概念和语法

(1)from子句

(2)where子句

(3)select子句

(4)group by 子句

(5) having 子句

(6) order by子句

3.Query接口取得和主要方法

【】Hibernate执行CUDR

1.执行C：insert

session.save(Model对象)

2.执行U：update

session.update(Model对象);

3.执行D：delete from

session.delete(Model对象)

4.执行R：select

(1)取得单个对象：session.get, session.load

(2)取得多个对象：

【】Hibernate查询机制类型

1.导航对象图方式：通过一个对象的属性可以取得另一个

Model对象。

DepartmentModel dm=session.get(DepartmentModel.class, departmentNo);

list.addAll(dm.getEmployees());

2. OID检索：通过对象的主键值取得对象

session.get, session.load

案例：

DepartmentModel dm=session.get(DepartmentModel.class, departmentNo);

3. HQL检索：通过HQL语言和Query接口实现

String hql="from EmployeeModel";

Query query=session.createQuery(hql);

List list=query.list();

4. QBC检索:通过Critetion接口，实现动态的条件。

5. SQL检索：执行针对特定数据库的SQL语句。

String sql="select \* from OA\_employee";

SQLQuery sqlquery=session.createSQLQuery(sql);

List list=query.list();

【】HQL概念

1. HQL- Hibernate Query Language

2.特点：针对Model对象的查询语言

SQL：select \* from OA\_Employee em

where age between 20 and 30

HQL: from EmployeeModel em

where em.age between 20 and 30

【】HQL语法

[select子句]

from子句

[where子句]

[group by子句]

[having 子句]

[order by 子句]

【】from子句

1.功能：指定检索的对象的类

2.语法：from Model类

from Model类 as 别名

from Model类 别名

from Model类 别名 inner join 别名.属性 别名

案例：

String hql="from EmployeeModel em";

【】where子句

1.功能：对检索的对象执行筛选。

2.语法：where 逻辑表达式

案例：

String hql="from EmployeeModel em

where em.age between 20 and 30";

3.常见的where表达式：

(1) 比较运算：>, >=, <, <=, =, !=, <>

String hql="from EmployeeModel em

where em.age>=20 and em.age<=30";

(2) 逻辑表达式：and or not

String hql="from EmployeeModel em

where em.age>=20 and em.age<=30";

(3)区间运算：between .. and .., not between and

String hql="from EmployeeModel em

where em.age not between 20 and 30";

(4) 模糊运算：like ..., not like ...

String hql="from EmployeeModel em

where em.name like '%国%' ";

(5)集合运算：in (....), not in(....)

String hql="from EmployeeModel em

where em.age in (20,30,40)";

(6)空对象运算：is null, is not null

String hql="from EmployeeModel em

where em.name is null";

(7)空集合运算：is empty, is not empty

String hql="from EmployeeModel em

where em.behaves is empty";

查询没有任何爱好的员工列表

【】select子句

1.功能：只查询类对象的部分属性。

2.语法：select 别名, 别名.属性,.....

3.案例：

(1)取得单个属性:

String hql=" select em.name from EmployeeModel em

where em.age not between 20 and 30";

Query query=session.createQuery(hql);

List<String> list=query.list();

(2)取得单个属性:

String hql=" select em.age from EmployeeModel em

where em.age not between 20 and 30";

Query query=session.createQuery(hql);

List<Integer> list=query.list();

(3)取得单个对象:

String hql=" select em from EmployeeModel em

where em.age not between 20 and 30";

Query query=session.createQuery(hql);

List<EmployeeModel> list=query.list();

(3)取得多个属性:

String hql=" select em.name, em.age,em.salary from EmployeeModel em

where em.age not between 20 and 30";

Query query=session.createQuery(hql);

List<Object[]> list=query.list();

【】Query接口取得

1.取得Query对象：

Query query=session.createQuery(hql);

【】Query接口方法

1.list() 执行HQL，返回对象列表

List<EmployeeModel> list=query.list();

2.setXxx(int position, Xxx value): 设定位置参数 ？

setXxx(String name, Xxx value): 设定命名参数 :name

Xxx类型：

案例1：使用位置参数？

String hql="from EmployeeModel em

where em.age between ? and ?";

Query query=session.createQuery(hql);

query.setInteger(0,minAge);

query.setInteger(1, maxAge);

案例2：使用命名参数参数:参数名

String hql="from EmployeeModel em

where em.age between :min and :max";

Query query=session.createQuery(hql);

query.setInteger("min",minAge);

query.setInteger("max", maxAge);

3.设定检索集合个数和位置方法

(1) setFirstResult(int)：设定结果的起始位置

(2)setMaxResults(int): 设定结果的的个数

public List<EmployeeModel> getListByDepartmentWithPage(int departmentNo, int rows, int page) throws Exception {

SessionFactory sf=HibernateFactory.getSessionFactory();

Session session=sf.openSession();

Transaction tx=session.beginTransaction();

//String hql="from EmployeeModel em where em.age between ? and ?";

String hql="from EmployeeModel em where em.department.no=:dno";

Query query=session.createQuery(hql);

query.setInteger("dno",departmentNo);

query.setFirstResult(rows\*(page-1));

query.setMaxResults(rows);

List<EmployeeModel> list=query.list();

tx.commit();

session.close();

return list;

}

第08讲 Hibernate高级查询

【】授课大纲

1.HQL内关联查询

2.HQL外关联查询

3.HQL分类汇总查询

4.HQL子查询

【】HQL内关联查询（inner join）

1.概念：多个Model对象内联是指双方

属性值对应，不能为null。

2.类型：

(1) 显式内关联

--查询每个员工姓名和部门名称

<1>

String hql="select em.name, dm.name

from EmployeeModel em inner join em.department dm";

<2>

String hql="select em.name, dm.name

from DepartmentModel dm inner join dm.employees em";

<3>

String hql="from DepartmentModel dm

inner join dm.employees em";

Query query=session.createQuery(hql);

List<Object[]> list=query.list();

<4>案例：关联所有的Model类

String hql="from BehaveModel bm inner join

bm.employees em inner join

em.department dm inner join

em.address am";

(2)隐式内关联

内关联中没有inner join语句。

只能适用于多对一，一对一。

//查询指定部门的员工列表，显式模式

String hql="select em from EmployeeModel em

inner join em.department dm

where dm.no=:dno";

//查询指定部门的员工列表，隐式模式

String hql="select em from EmployeeModel em

where em.department.no=:dno";

//查询指定爱好的员工列表，隐式模式 （X）

String hql="select em from EmployeeModel em

where em.behaves.no=:bno";

//查询指定爱好的员工列表，显式模式 （V）

String hql="select em from EmployeeModel em

inner join em.behaves bm

where bm.no=:bno";

(3)抓取内关联

直接取得关联的对象。

String hql="from DepartmentModel dm inner join fetch

dm.employees em where dm.no=:dno";

【】HQL外关联查询（outer join）

1.概念：可以查询没有对应的对象。

2.类型：左外关联，右外关联，

3.语法与案例

(1) 左外联

hql="from DepartmentModel dm

left outer join dm.employees em"

(1) 右外联

hql="from DepartmentModel dm

right outer join dm.employees em"

【】交叉关联

hql="from DepartmentModel dm, EmployeeModel em";

【】HQL分类汇总查询

1.相关的语句和函数

(1) 函数：

count(), min(), max(), avg(), sum()

(2) 语句：

group by 分组 , having 对汇总结果再筛选

2.案例：

(1) 取得所有员工的平均工资

hql="select avg(em.salary) from EmployeeModel em";

Query query=session.createQuery(hql);

Double avgsal=(Double)query.uniqueResult();

(2)取得每个部门的员工的平均工资

hql="select em.department.name,avg(em.salary) from EmployeeModel em group by em.department.name";

Query query=session.createQuery(hql);

List<Object[]> list=query.list();

(3)取得每个部门的员工的平均工资大于3000的。

hql="select em.department.name,avg(em.salary) from EmployeeModel em

group by em.department.name

having avg(em.salary)>3000

";

Query query=session.createQuery(hql);

List<Object[]> list=query.list();

【】HQL子查询

1.HQL支持在where和having使用子查询

2.案例：

(1)没有爱好的员工列表

hql=" from EmployeeModel emm where emm not in

(select em from EmployeeModel em

inner join em.behaves)"

hql="select em from EmployeeModel em

left outer join em.behaves bm

where bm is null";

【】补充：Query的取得查询结果的方法

1.取集合方法

List list=query.list();

2.单个结果方法：

Object o=query.uniqueResult();

【】补充：SQL外关联查询

--查询每个员工姓名和部门名称

select a.empname,b.deptname

from OA\_Employee a

left outer join

OA\_Department b

on a.deptno=b.deptno

union

select a.empname,b.deptname

from OA\_Employee a

right outer join

OA\_Department b

on a.deptno=b.deptno

第09讲 Spring概述

【】授课大纲

1.Spring概念

2.Spring功能

3.Spring组成

4.Spring引入与配置

5.IoC与DI概念

6.Spring Bean概念

7.Spring Bean的配置

8.DI类型与配置

【】Spring概念

Hibernate是javaEE的持久层框架。

Struts2是JavaEE的VC层框架。

Spring是JavaEE应用的各层对象的管理框架。

【】Spring功能

1.负责各层对象的创建和销毁。

2.负责各层对象的依赖对象的管理。

【】Spring组成

1.核心容器(IoC Container)

(1)core模块

(2)Beans模块

(3)Context模块

(4)SPEL模块：

2.AOP模块：

3.AspectJ模块

4.Instrumentation模块

5.Message模块

6.数据层集成模块

(1) JDBC模块

(2)ORM模块：管理持久层对象的模块

(3)OXM模块：

(4)JMS模块：

(5) Transaction模块：事务处理模块。

7.Web模块

(1) Web MVC：Web框架

(2)WebSocket模块：

(3)Portlet：门户模块

(4)Sevlet模块：

8.Testing模块

【】Spring引入与配置

1.引入Spring JAR文件

2.创建Spring的配置文件：

(1)文件名：context.xml, bean.xml, applicationContext.xml

(2)位置：classpath根目录

(3)内容：

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

<beans xmlns="http://www.springframework.org/schema/beans"

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

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:c="http://www.springframework.org/schema/c"

xmlns:cache="http://www.springframework.org/schema/cache"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:jdbc="http://www.springframework.org/schema/jdbc"

xmlns:jee="http://www.springframework.org/schema/jee"

xmlns:jms="http://www.springframework.org/schema/jms"

xmlns:lang="http://www.springframework.org/schema/lang"

xmlns:mvc="http://www.springframework.org/schema/mvc"

xmlns:oxm="http://www.springframework.org/schema/oxm"

xmlns:p="http://www.springframework.org/schema/p"

xmlns:task="http://www.springframework.org/schema/task"

xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:util="http://www.springframework.org/schema/util"

xmlns:websocket="http://www.springframework.org/schema/websocket"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.3.xsd

http://www.springframework.org/schema/cache http://www.springframework.org/schema/cache/spring-cache-4.3.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.3.xsd

http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc-4.3.xsd

http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-4.3.xsd

http://www.springframework.org/schema/jms http://www.springframework.org/schema/jms/spring-jms-4.3.xsd

http://www.springframework.org/schema/lang http://www.springframework.org/schema/lang/spring-lang-4.3.xsd

http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.3.xsd

http://www.springframework.org/schema/oxm http://www.springframework.org/schema/oxm/spring-oxm-4.3.xsd

http://www.springframework.org/schema/task http://www.springframework.org/schema/task/spring-task-4.3.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.3.xsd

http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.3.xsd

http://www.springframework.org/schema/websocket http://www.springframework.org/schema/websocket/spring-websocket-4.3.xsd">

<!-- Spring Bean的配置代码 -->

</beans>

【】补充1：Java编程的核心

1.编写class类。

2.创建类对象。

3.调用对象的方法。

【】补充2：Java取得对象的方式

1.直接主动创建型：

A a=new A();

2.间接主动创建型

A a=Factory.createA();

3.间接主动查找型

A a=(A)ctx.lookup("a");

4.间接被动注入型（IOC型）

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{}

【】IoC与DI概念

1.IoC：反向控制

(1) IoC：Inversion of Control

(2) 使用的对象有其他对象注入。

2.DI：依赖注入

(1) DI：Dependency Injection

(2) 依赖的对象是其他对象注入的。

【】Spring Bean概念

Spring管理的类对象，称为Spring Bean。

【】Spring Bean的配置

1.Spring创建对象方式1：使用new 构造方法

语法：

<bean id="" class="包名.类名" />

或

<bean id="" class="" >

</bean>

案例：

<bean id="deparmentService"

class="com.city.oa.service.impl.DepartmentServiceImpl">

</bean>

2.Spring创建对象方式2：使用Factory的静态方法

语法：

<bean id="" class="包名.工厂类名" factory-method="静态方法名" />

或

<bean id="" class="包名.工厂类名" factory-method="静态方法名" >

</bean>

案例：

<bean id="sf" class="com.city.oa.factory.HibernateFactory"

factory-method="getSessionFactory"

>

</bean>

【】取得Spring管理的Bean对象

1.连接到Spring的IoC容器：

ApplicationContext ac=new ClassPathXmlApplicationContext("context.xml");

2.取得Spring管理的对象

//ds=(IDepartmentService)ac.getBean("deparmentService");

ds=ac.getBean("deparmentService",IDepartmentService.class);

【】Spring DI注入实现的方式

1.属性注入：

(1)定义属性变量

private SessionFactory sf=null;

(2)生成set方法

public void setSf(SessionFactory sf) {

this.sf = sf;

}

(3)配置set注入：

<bean id="sf000"

class="com.city.oa.factory.HibernateFactory"

factory-method="getSessionFactory"

></bean>

<!-- 创建部门的业务对象 -->

<bean id="deparmentService"

class="com.city.oa.service.impl.DepartmentServiceImpl">

<property name="sf" ref="sf000" />

</bean>

2.构造方法注入：

【】DI类型与配置

第10讲 Spring AOP编程（第14章）

【】授课大纲

1.AOP概念

2.AOP功能

3.AOP实现技术

4.AOP实现方式

5.Spring AOP组成元素

6.Spring AOP Advice类型

7.Spring AOP实现方式

(1) Advice接口实现类+XML配置

(2) POJO类+XML配置

(3) POJO类+Java注释

【】AOP概念

1. AOP：Aspect Oriented Programming

面向切面编程

OOP: Object Oriented Programming

Aspect ：切面是指多个对象的多个方法中重复的代码。

【】AOP功能

实现多个方法内相同代码的重用。

【】代码重用主要机制

1.OOP的继承

2.AOP的切面

3.工厂模式

【】AOP实现技术

1.JavaEE的Filter

2.Struts2的拦截器

3.Spring AOP

4.JBoss AOP

5.AspectJ框架：

【】AOP实现方式

1.静态AOP实现：在编译阶段，将切面类与目标类编译成

一个.class文件。

2.动态AOP实现：在运行时，将切面对象注入到目标对象。

Spring AOP只支持动态AOP。

【】Spring AOP组成元素

1.连接点：joinpoint

切面代码运行的地点，Spring只支持方法连接点。

2.切入点：Pointcut

符合一定规律的连接点的集合。

3.通知Advice

切面的运行代码。

4.切面(Aspect)

通知Advice+切入点Pointcut=切面(Aspect)

5.目标对象（Target）

切面代码切入的对象。

6.代理对象Proxy

代理对象Proxy=目标对象（Target）+切面(Aspect)

Spring自动生成包含切面和目标对象的代理对象。

7.织入Weaving

把切面与目标对象集成的过程。

(1) XML配置

(2) Java注释配置

(3)编程方式

【】Spring AOP 切面类型

1.方法前切面：

目标对象方法前切入。

2.方法后切面：

方法结束，可以有异常。

3.方法返回后切面：

方法正常结束，没有异常。

4.方法异常切面：

方法执行出现异常时切入。

5. 方法环绕切面：

方法前，后都切入。

【】Spring AOP实现方式

(1) Advice接口实现类+XML配置

(2) POJO类+XML配置

(3) POJO类+Java注释

【】Advice接口实现类编程

1.方法前Advice：

(1)实现接口：MethodBeforeAdvice

public class ServiveMethodBeforeAdviceWithInterface implements MethodBeforeAdvice {

@Override

public void before(Method method, Object[] args, Object target) throws Throwable {

System.out.println("方法前Advice，拦截的方法名："+method.getName()+",切入的类是:"+target.getClass().getName());

}

}

【】切面的XML配置

1.声明切面类

<bean id="" class="" />

HS第12讲 Spring集成Hibernate

【】授课内容

1.Spring集成Hibernate的基本原理

2.Spring集成Hibernate的基本步骤

3.Spring配置DataSource

4.Spring配置SessionFactory

5.Spring配置事务管理器

6.Spring配置事务Advice

7.Spring配置事务切面

【】HibernateSpring课程作业：

1.作业1,2,3

2.位置：ftp://210.30.108.30

3.要求：手写

4.提交：12周周五之前

【】Spring集成Hibernate的基本原理

1.Spring获取SessionFactory对象。

2.Spring创建事务管理器

3.Spring创建事务Advice

4.Spring管理切入点。

5.Spring管理事务切面。

【】Spring获取SessionFactory对象

1.配置数据源：连接数据库

(1)使用Spring内置的连接池对象

<bean id="galleryDataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource"

p:driverClassName="driverClassName"

p:url="url"

p:username="username"

p:password="password" />

(2)使用JavaEE Server配置的连接池

<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean" >

<property name="jndiName">

<value>java:comp/env/cityoa</value>

</property>

</bean>

(3)使用第三方连接池框架(C3P0, DBCP, Proxocol)

DBCP配置：P542 代码18.7

C3P0配置：

<bean id="" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">

<property name="driverClass" value="com.mysql.jdbc.Driver"></property>

<property name="jdbcUrl" value="jdbc:mysql://localhost:3306/cityoa"></property>

<property name="user" value="root"></property>

<property name="password" value="root"></property>

<property name="acquireIncrement" value="1"></property>

<property name="initialPoolSize" value="1"></property>

<property name="maxPoolSize" value="2"></property>

<property name="minPoolSize" value="1"></property>

<property name="acquireRetryDelay" value="100"></property>

<property name="acquireRetryAttempts" value="10"></property>

<property name="breakAfterAcquireFailure" value="false"></property>

</bean>

2.配置Hibernate SessionFactory

Spring支持不同版本的Hibernate。

...hibernate3.LocalSessionFactoryBean

...hibernate4.LocalSessionFactoryBean

...hibernate5.LocalSessionFactoryBean

<beanid="sessionFactory" class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">

<property name="dataSource" ref="dataSource"/>

<property name="mappingDirectoryLocations">

<list>

<value>classpath:/com/city/oa/value/</value>

</list>

</property>

<property name="hibernateProperties">

<props>

<prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>

<prop key="hibernate.max\_fetch\_depth">3</prop>

<prop key="hibernate.jdbc.fetch\_size">50</prop>

<prop key="hibernate.jdbc.batch\_size">10</prop>

<prop key="hibernate.show\_sql">true</prop>

</props>

</property>

</bean>

注：

指定XML映射文件的查找：

目录方式

<property name="mappingDirectoryLocations">

<list>

<value>classpath:/com/city/oa/model/</value>

</list>

</property>

文件方式：P543 代码18.8

指定扫描的注释方式的映射类：

包位置方式：扫描包中的类有@Entiry的类。

<property name="packagesToScan">

<list>

<value>com.city.oa.model</value>

</list>

</property>

映射类文件方式：

【】配置Spring管理Hibernate的事务管理器

<bean id="transactionManager" class="org.springframework.orm.hibernate5.HibernateTransactionManager">

<property name="sessionFactory" ref="sessionFactory"/>

</bean>

【】配置事务的Advice

1.XML方式：

<tx:advice id="txAdvice" >

<tx:attributes>

<tx:method name="get\*" read-only="true" rollback-for="Exception"/>

<tx:method name="\*" rollback-for="Exception"/>

</tx:attributes>

</tx:advice>

<!-- 将事务Advice与业务方法结合 -->

<aop:config>

<aop:pointcut id="servicepointcut" expression="execution(\* com.city.oa.service.impl.\*.\*(..))"/>

<aop:advisor pointcut-ref="servicepointcut" advice-ref="txAdvice"/>

</aop:config>

2.注释方式：

(1)在业务类或业务方法上增加事务注释

@Transactional

(2)在Spring配置文件里增加启用事务注释功能：

<tx:annotation-driven/>

<tx:annotation-driven transaction-manager="transactionManager" />

【】

模板使用Bootstrap框架

【】Spring管理事务下的Hibernate的编程

1.注入SessionFactory

2.取得Spring事务Advice管理下的Session

Spring集成Hibernate辅助资料

【】Tomcat配置数据库连接池

1.文件名称:context.xml

2.位置：\tomcat\conf\context.xml

3.配置语法：

<Resource

name="cityoa"

auth="Container"

type="javax.sql.DataSource"

driverClassName="com.mysql.jdbc.Driver"

maxIdle="2"

maxWait="5000"

url="jdbc:mysql://localhost:3306/cityoa"

username="root"

password="root"

maxActive="20"

removeAbandoned="true"

removeAbandonedTimeOut="5"

logAbandoned="true"

/>

【】Spring中配置查找Tomcat配置的连接池

1.使用JndiFactoryBean

<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean" >

<property name="jndiName">

<value>java:comp/env/cityoa</value>

</property>

</bean>

2.使用<jee>命名空间的语法

<jee:jndi-lookup id="dataSource01" jndi-name="java:comp/env/cityoa"></jee:jndi-lookup>

【】Spring自主配置数据库连接池

Spring框架内置了数据库连接池管理类 DriverManagerDataSource

<bean id="galleryDataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

p:driverClassName="driverClassName"

p:url="url"

p:username="username"

p:password="password"/>

</bean>

此种模式一般用于开发测试阶段，实际部署时推荐使用应用服务器管理的连接池方式。

【】使用DBCP连接池管理框架

<beanid="galleryDataSource" class="org.apache.commons.dbcp.BasicDataSource">

p:driverClassName="driverClassName"

p:url="url"

p:username="username"

p:password="password"/>

</bean>

【】使用C3P0数据库连接池管理框架

<beanid="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">

<propertyname="driverClass" value="com.mysql.jdbc.Driver"></property>

<propertyname="jdbcUrl" value="jdbc:mysql://10.25.176.250:3306/htdb"></property>

<property name="user" value="root"></property>

<property name="password" value="root123"></property>

<property name="acquireIncrement" value="2"></property>

<property name="initialPoolSize" value="3"></property>

<property name="maxPoolSize" value="5"></property>

<property name="minPoolSize" value="1"></property>

<property name="acquireRetryDelay" value="100"></property>

<property name="acquireRetryAttempts" value="10"></property>

<property name="breakAfterAcquireFailure" value="false"></property>

</bean>

【】Proxcol连接池配置

<beanid="proxoolDataSource" class="org.logicalcobwebs.proxool.ProxoolDataSource">

<propertyname="driver" value="${jdbc.connection.driverClassName}"/>

<propertyname="driverUrl" value="${jdbc.connection.url}"/>

<propertyname="user" value="${jdbc.connection.username}"/>

<propertyname="password" value="${jdbc.connection.password}"/>

<!-- 测试的SQL执行语句 -->

<propertyname="houseKeepingTestSql" value="${proxool.houseKeepingTestSql}"/>

<!-- 最少保持的空闲连接数 （默认2个） -->

<propertyname="prototypeCount" value="${proxool.prototypeCount}"/>

<!-- proxool自动侦察各个连接状态的时间间隔(毫秒),侦察到空闲的连接就马上回收,超时的销毁 默认30秒） -->

<propertyname="houseKeepingSleepTime" value="${proxool.hourseKeepingSleepTime}"/>

<!-- 最大活动时间(超过此时间线程将被kill,默认为5分钟) -->

<propertyname="maximumActiveTime" value="${proxool.maximumActiveTime}"/>

<!-- 连接最长时间(默认为4个小时) -->

<propertyname="maximumConnectionLifetime" value="${proxool.maximumConnectionLifetime}"/>

<!-- 最小连接数 （默认2个） -->

<propertyname="minimumConnectionCount" value="${proxool.minimumConnectionCount}"/>

<!-- 最大连接数 （默认5个） -->

<propertyname="maximumConnectionCount" value="${proxool.maximumConnectionCount}"/>

<!-- -->

<propertyname="statistics" value="${proxool.statistics}"/>

<!-- 别名 -->

<property name="alias" value="${proxool.alias}"/>

<!-- -->

<propertyname="simultaneousBuildThrottle" value="${proxool.simultaneous-build-throttle}"/>

</bean>

然后注入到sessionFactory中

<beanid="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

<property name="dataSource" ref="proxoolDataSource"/>

</bean>

【】Spring配置Hibernate的SessionFactory

1.Spring自己配置Hibernate配置信息，不再需要hibernate.cfg.xml文件。

<beanid="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">

<property name="dataSource" ref="dataSource"/>

<property name="mappingDirectoryLocations">

<list>

<value>classpath:/com/city/oa/value/</value>

</list>

</property>

<property name="hibernateProperties">

<props>

<prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>

<prop key="hibernate.max\_fetch\_depth">3</prop>

<prop key="hibernate.jdbc.fetch\_size">50</prop>

<prop key="hibernate.jdbc.batch\_size">10</prop>

<prop key="hibernate.show\_sql">true</prop>

</props>

</property>

</bean>

2.Spring依靠hibernate.cfg.xml文件，取得Hibernate的配置信息

<beanid="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">

<property name="dataSource" ref="dataSource" />

<property name="configLocation" value="classpath:hibernate.cfg.xml" />

</bean>

【】LocalSessionFactoryBean取得Hibernate映射信息的属性配置

1. 属性mappingDirectoryLocations 指定映射文件的目录，自动扫描以.hbm.xml结尾的文件。

<property name="mappingDirectoryLocations">

<list>

<value>classpath:/com/city/oa/value/</value>

</list>

</property>

2.属性：mappingLocations 指定映射文件

(1)可以指定具体的映射文件

<property name="mappingLocations">

<list>

<value>classpath:com/apress/springrecipes/course/Course.hbm.xml</value>

</list>

</property>

(2)使用匹配符指定映射文件

<propertyname="mappingLocations" value="classpath:com/apress/springrecipes/course/\*.hbm.xml" />

3.属性：mappingResources指定单独的映射文件

<property name="mappingResources">

<list>

<value>com/apress/springrecipes/course/Course.hbm.xml</value>

<value>com/apress/springrecipes/course/Student.hbm.xml</value>

</list>

</property>

4.指定扫描的注释类的路径：

<property name="packagesToScan">com.neusoft.oa.value</property>

或：

<property name="packagesToScan">

<list>

<value>com.geek.core.pojo</value>

</list>

</property>

<beanid="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">

<!--指定Hibernate核心配置文件 数据库配置信息在hibernate.cfg.xml内-->

<propertyname="configLocation" value="/WEB-INF/hibernate.cfg.xml"/>

<propertyname="packagesToScan" value="com.springapp.mvc.entity.\*" />

</bean>

5.

【】配置Hibernate的事务管理器

<beanid="transactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">

<property name="sessionFactory" ref="sessionFactory"/>

</bean>

【】Spring1.x配置事务管理器

<beanid="departmentBusiness" class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">

 <!--   为事务代理bean注入事务管理器-->

<propertyname="transactionManager"><refbean="transactionManager"/></property>

<!--   设置事务属性-->

 <property name="transactionAttributes">

<props>

    <!--   所有以find开头的方法，采用required的事务策略，并且只读-->

   <prop key="find\*">PROPAGATION\_REQUIRED,readOnly</prop>

    <!--   其他方法，采用required的事务策略 ->

     <prop key="\*">PROPAGATION\_REQUIRED</prop>

   </props>

  </property>

  <!--   为事务代理bean设置目标bean -->

 <property name="target">

    <ref local="departmentBusinessTarget"/>

 </property>

 </bean>

【】Spring2.x配置Spring的事务Advice

1.使用默认的事务管理器，如果存在id为transactionManager的事务管理器bena，可以不用指定

<!-- 事务处理Advice -->

<tx:advice id="txAdvice" >

<tx:attributes>

<tx:method name="get\*" read-only="true" rollback-for="Exception"/>

<tx:method name="getList\*" propagation="NEVER"/>

<tx:method name="\*" rollback-for="Exception"/>

</tx:attributes>

</tx:advice>

2.配置Spring的事务Advice，通过属性transaction-manager指定事务管理器。

<!-- 配置事务通知属性 -->

<tx:advice id="txAdvice" transaction-manager="transactionManager">

<!-- 定义事务传播属性 -->

<tx:attributes>

<tx:method name="add\*" propagation="REQUIRED" />

<tx:method name="modify\*" propagation="REQUIRED" />

<tx:method name="delete\*" propagation="REQUIRED" />

<tx:method name="change\*" propagation="REQUIRED" />

<tx:method name="update\*" propagation="REQUIRED" />

<tx:method name="new\*" propagation="REQUIRED" />

<tx:method name="set\*" propagation="REQUIRED" />

<tx:method name="remove\*" propagation="REQUIRED" />

<tx:method name="get\*" propagation="REQUIRED" read-only="true" />

<tx:method name="find\*" propagation="REQUIRED" read-only="true" />

<tx:method name="load\*" propagation="REQUIRED" read-only="true" />

<tx:method name="\*" propagation="REQUIRED" read-only="true" />

</tx:attributes>

</tx:advice>

【】配置事务切入点和切面

<aop:config>

<aop:pointcut id="businessImpl" expression="execution(\* com.city.oa.business.impl.\*Impl\*.\*(..))"/>

<aop:advisor pointcut-ref="businessImpl" advice-ref="txAdvice"/>

</aop:config>

【】常用的AspectJ切入点表达式

(1)execution(\* com.spring.service.\*.\*(..))

表示com.spring.service包下，返回值为任意类型；方法名任意；参数任意的所有方法

(2)execution(\* com.neusoft.oa.business.impl.\*Impl\*.\*(..))

(3)execution(\* void com.neusoft.oa.business.impl.\*Impl\*.\*(..))

(4) execution(public \* com.city.oa.business.impl.\*Impl\*.add\*(..))

【】Spring与Web Application集成

JavaEE Web应用可启动时自动启动Spring ApplicationContext容器。

在/web-inf/web.xml中配置

1.查找Spring配置文件的位置或文件

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath\*:META-INF/spring/spring-master.xml</param-value>

</context-param>

2.配置Spring监听器

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

Spring事务管理辅助资料

【】Spring XML配置

1.

<beans xmlns="http://www.springframework.org/schema/beans"

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

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop.xsd">

<context:annotation-config/>

</beans>

【】Spring事务管理器基础类

org.springframework.transaction.PlatformTransactionManager

所有具体平台下的事务管理器都是其子类型。，包括

1.JDBC事务管理器

2.hibernate事务管理器

3.JPA事务管理器

4.JDO事务管理器

5.TopLink事务管理器

【】Spring的事务管理器类型（Transaction Manager Type）

1.JDBC的事务管理器

org.springframework.jdbc.datasource.DataSourceTransactionManager

JDBC事务管理器配置案例：

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">

<property name="driverClassName" value="${jdbc.driverClassName}" />

<property name="url" value="${jdbc.url}" />

<property name="username" value="${jdbc.username}" />

<property name="password" value="${jdbc.password}" />

</bean>

<bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="dataSource"/>

</bean>

JDBC事务管理器需要dataSource属性注入。

2.Hibernate事务管理器

org.springframework.orm.hibernate.HibernateTransactionManager

org.springframework.orm.hibernate3.HibernateTransactionManager

org.springframework.orm.hibernate4.HibernateTransactionManager

org.springframework.orm.hibernate5.HibernateTransactionManager

Hibernate事务管理器配置案例

<bean id="sessionFactory" class="org.springframework.orm.hibernate4(3).LocalSessionFactoryBean">

<property name="dataSource" ref="dataSource" />

<property name="mappingResources">

<list>

<value>org/springframework/samples/petclinic/hibernate/petclinic.hbm.xml</value>

</list>

</property>

<property name="hibernateProperties">

<value>

hibernate.dialect=${hibernate.dialect}

</value>

</property>

</bean>

<bean id="txManager" class="org.springframework.orm.hibernate4(3).HibernateTransactionManager">

<property name="sessionFactory" ref="sessionFactory" />

</bean>

3.JTA事务管理器

org.springframework.jta.JtaTransactionManager

JTA事务管理器配置案例：

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

<beans xmlns="http://www.springframework.org/schema/beans"

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

xmlns:jee="http://www.springframework.org/schema/jee"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/jee

http://www.springframework.org/schema/jee/spring-jee.xsd">

<jee:jndi-lookup id="dataSource" jndi-name="jdbc/jpetstore"/>

<bean id="txManager" class="org.springframework.transaction.jta.JtaTransactionManager" />

<!-- other <bean/> definitions here -->

</beans>

4.JPA事务管理器

org.springframework.orm.jpa.JpaTransactionManager

5.JDO事务管理器

org.springframework.orm.jdo.JdoTransactionManager

6. TopLink事务管理器

org.springframework.orm.toplink.TopLinkTransactionManager

【】XML方式声明事务管理器

1.Hibernate3的事务管理器声明：

<beanid="dataSource" class="org.apache.commons.dbcp.BasicDataSource">

<property name="driverClassName">

<value>org.hsqldb.jdbcDriver</value>

</property>

<property name="url">

<value>jdbc:hsqldb:hsql://localhost/hiberdb</value>

</property>

<property name="username">

<value>sa</value>

</property>

<property name="password">

<value></value>

</property>

</bean>

<beanid="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

<property name="mappingResources">

<list>

<value>com/packtpub/springhibernate/ch12/Student.hbm.xml</value>

<value>com/packtpub/springhibernate/ch12/Teacher.hbm.xml</value>

<!-- Other hbm files -->

</list>

</property>

<property name="hibernateProperties">

<props>

<prop key="hibernate.transaction.factory\_class">

org.hibernate.transaction.JDBCTransactionFactory

</prop>

<prop key="hibernate.dialect">org.hibernate.dialect.HSQLDialect</prop>

</props>

</property>

<property name="dataSource">

<ref local="dataSource"/>

</property>

</bean>

<beanid="transactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">

<property name="sessionFactory">

<ref local="sessionFactory"/>

</property>

</bean>

【】Spring1.x事务处理声明XML模式

1.先声明事务拦截器

<bean id="transactionInterceptor" class="org.springframework.transaction.interceptor.TransactionInterceptor">

<property name="transactionManager" ref="transactionManager"/>

<property name="transactionAttributes">

<props>

<prop key="\*">PROPAGATION\_REQUIRED</prop>

</props>

</property>

</bean>

通常情况下，当拦截的方法抛出非检查异常时，自动rollback事务。如果希望抛出检查异常时也自动回滚事务，需要增加如下配置

<bean id="transactionInterceptor" class="org.springframework.transaction.interceptor.TransactionInterceptor">

<property name="transactionManager" ref="transactionManager"/>

<property name="transactionAttributes">

<props>

<prop key="\*">PROPAGATION\_REQUIRED, -Throwable</prop>

</props>

</property>

</bean>

2.声明事务拦截器类（一般是业务实现类）的代理类

<bean id="transactionalStudentDao" class="org.springframework.aop.framework.ProxyFactoryBean">

<property name="target">

<ref local="studentDao"/>

</property>

<property name="interceptorNames">

<list>

<idref bean="transactionInterceptor"/>

</list>

</property>

</bean>

3.使用自动代理类，为指定的业务对象自动生成代理类对象

<bean class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">

<property name="beanNames">

<list>

<idref bean="studentDao"/>

</list>

</property>

<property name="interceptorNames">

<list>

<idref bean="transactionInterceptor"/>

</list>

</property>

</bean>

此时不再需要取得代理类对象，去调用业务方法，而是直接获得目标对象，调用其方法时，Spring自动创建代理类对象。

通过属性beanNames的list标记同时为多个业务目标对象增加自动代理类。

4.可以直接使用TransactionProxyFactoryBean，不需要声明TransactionInterceptor

<bean id="transactionalStudentDao" class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">

<property name="transactionManager">

<ref bean="transactionManager"/>

</property>

<property name="target">

<ref local="studentDao"/>

</property>

<property name="transactionAttributes">

<props>

<prop key="save\*">PROPAGATION\_REQUIRED</prop>

</props>

</property>

</bean>

【】Spring2.x+ 事务处理声明XML模式

1.Spring2.x增加了事务的命名空间<tx>，首先需要在Spring IoC配置文件中增加tx空间的声明

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

<beans xmlns="http://www.springframework.org/schema/beans"

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

xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx.xsd">

</beans>

2.<tx>命名空间提供了如下标记

(1)<tx:annotation-driven/>

<tx:annotation-driven transaction-manager="transactionManager" />

启动声明式事务处理，当目标对象的类或方法有事务注释@Transactional时，自动处理事务。

(2)<tx:advice>

声明事务Advice

<tx:advice id="txAdvice" transaction-manager="txManager">

<!-- the transactional semantics... -->

<tx:attributes>

<!-- all methods starting with 'get' are read-only -->

<tx:method name="get\*" read-only="true"/>

<!-- other methods use the default transaction settings (see below) -->

<tx:method name="\*"/>

</tx:attributes>

</tx:advice>

(3)<tx:attributes>

对事务Advice的属性进行配置，包括隔离级别，事务传播，超时时间。

(4)<tx:method>

配置事务的方法

3.事务声明处理的案例

<bean id="dataSource" class="org.springframework.jndi.

JndiObjectFactoryBean">

<property name="jndiName">

<value>java:comp/env/jdbc/MyDB</value>

</property>

</bean>

<bean id="sessionFactory"

class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

<property name="mappingResources">

<list>

<value>com/packtpub/springhibernate/ch12/Student.hbm.xml</value>

<value>com/packtpub/springhibernate/ch12/Teacher.hbm.xml</value>

<value>com/packtpub/springhibernate/ch12/Course.hbm.xml</value>

</list>

</property>

<property name="hibernateProperties">

<props>

<prop key="hibernate.dialect">

org.hibernate.dialect.HSQLDialect

</prop>

</props>

</property>

<property name="dataSource">

<ref local="dataSource"/>

</property>

</bean>

<bean id="transactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">

<property name="sessionFactory">

<ref local="sessionFactory"/>

</property>

</bean>

<!-- Transactional proxy -->

<tx:advice id="transactionInterceptor" transaction-manager="transactionManager">

<tx:attributes>

<tx:method name="\*" rollback-for="Throwable"/>

</tx:attributes>

</tx:advice>

当发生指定的异常时，启动自动回滚的配置：

<tx:advice id="txAdvice" transaction-manager="txManager">

<tx:attributes>

<tx:method name="get\*" read-only="true" rollback-for="Exception"/>

<tx:method name="\*"/>

</tx:attributes>

</tx:advice>

以上配置则表明，业务方法抛出任何异常时，都会自动回滚。

如果要排除指定异常时，不自动回滚，则配置代码如下：

<tx:advice id="txAdvice">

<tx:attributes>

<tx:method name="updateStock" no-rollback-for="InstrumentNotFoundException"/>

<tx:method name="\*"/>

</tx:attributes>

</tx:advice>

将回滚和不回滚的异常配置在一起：

<tx:advice id="txAdvice">

<tx:attributes>

<tx:method name="\*" rollback-for="Throwable" no-rollback-for="InstrumentNotFoundException"/>

</tx:attributes>

</tx:advice>

【】事务AOP切入点配置：

1.

在定义advisor时，直接定义pointcut属性定义切入点，再通过advice-ref引入Advice：

<aop:config>

<aop:advisor pointcut=" execution(\* com.packtpub.springhibernate.ch12.\*Service.\*(..))" advice-ref="txAdvice"/>

</aop:config>

2.

或单独使用<aop:pointcut>定义切入点，再定义advisor,组装切入点和Advice.

<aop:config>

<aop:pointcut id="fooServiceOperation" expression="execution(\* x.y.service.FooService.\*(..))"/>

<aop:advisor advice-ref="txAdvice" pointcut-ref="fooServiceOperation"/>

</aop:config>

3.

推荐将业务层的所有方法都切入事务Advice，配置代码如下：

<aop:config>

<aop:pointcut id="businessMethods" expression="execution(\* com.neusoft.oa.business.impl.\*.\*(..))"/>

<aop:advisor advice-ref="txAdvice" pointcut-ref="businessMethods"/>

</aop:config>

======================================================================================================

【】使用Java Annotation注释方式的事务

1.配置Spring使用注释模式的事务声明

在Spring IoC的配置文件中加入;

使用默认名称的事务管理器，此时事务管理类命名为transactionManager。

<tx:annotation-driven/>

如果事务管理类bean的名称不是transactionManager,则需要指定transaction-manager属性

来指定事务管理器：

<tx:annotation-driven transaction-manager="txManager"/>

Spring框架将自动扫描类中有事务声明的类或方法

2.类事务声明

@Service("behaveBusiness")

@Transactional

public class BehaveBusinessImpl implements IBehaveBusiness {

}

如果申明为只读到事务，则使用代码

@Transactional(readOnly = true)

public class BehaveBusinessImpl implements IBehaveBusiness {

}

3.方法的事务声明

public class BehaveBusinessImpl implements IBehaveBusiness {

@Transactional

public void add() throws Exception{

}

@Transactional(readOnly=true)

public List getListByAll() throws Exception{

}

}

4.事务声明的案例：

@Transactional(readOnly = true)

public class DefaultFooService implements FooService {

public Foo getFoo(String fooName) {

// do something

}

// these settings have precedence for this method

@Transactional(readOnly = false, propagation = Propagation.REQUIRES\_NEW)

public void updateFoo(Foo foo) {

// do something

}

}

【】@Transactional事务属性

1.readOnly

取值true或false，默认false

@Transactional(readOnly = true)

2.propagation

@Transactional(propagation = Propagation.REQUIRES\_NEW)

Spring旧版本的取值：

Propagation.PROPAGATION\_REQUIRED

Propagation.PROPAGATION\_SUPPORTS

Propagation.PROPAGATION\_MANDATORY

Propagation.PROPAGATION\_REQUIRES\_NEW

Propagation.PROPAGATION\_NOT\_SUPPORTED

Propagation.PROPAGATION\_NEVER

Propagation.PROPAGATION\_NESTED

Spring新版本的取值：

Propagation.NEVER

Propagation.NOT\_SUPPORTED

Propagation.

3.isolation

取值：

TRANSACTION\_NONE

TRANSACTION\_READ\_UNCOMMITTED

TRANSACTION\_READ\_COMMITTED

TRANSACTION\_REPEATABLE\_READ

TRANSACTION\_SERIALIZABLE

4.rollbackFor 事务回滚配置

@Transactional(rollbackFor=java.lang.Exception.class)

或：

@Transactional(rollbackFor=Exception.class)

5.rollbackForClassName

6.noRollbackFor

@Transactional(notRollbackFor=RunTimeException.class)

7.noRollbackForClassName

8.事务配置案例：

@Transactional(rollbackFor=InvalidImageExeption.class,

readOnly=false,

timeout=30,

propagation=Propagation.SUPPORTS,

isolation=Isolation.DEFAULT)

public void saveArtEntity(ArtEntity artEntity) throws InvalidImageException {

this.getArtEntityDao().saveArtEntity(artEntity);

}