**中软国际考题-笔试**

1. 单选题
2. 在Spring中为了方便使用JNDI获取数据源连接，还给我们提供了(A)类。

A．org.springframework.jndi.JndiObjectFactoryBean

B. org.springframework.cache.jcache.config

C. org.springframework.cache.jcache

D. org.springframework.jdbc.datasource.SimpleDriverDataSource

2、query()方法的返回值是（C）

A. bool类型

B. Map集合

C. List集合

D. String类型

3、不属于core核心包的是（B）

A. JdbcTemplate类

B. SQLException类

C. SimpleJdbcInsert类

D. SimpleJdbcCall类

4、下列不属于execute()方法作用的是（A）

A. 执行SQL语句的查询功能

B. 执行数据表的创建功能

C. 执行数据表的删除功能

D. 执行数据的更新功能

5、JDBC中负责建立与数据库连接的类是（ D ）

A.Statement

B.PreparedStatement

 C.ResultSet

D.DriverManager

6.在Java Web应用程序中，使用哪个数据源与JNDI获得数据库连接时，应把JDBC驱动程序jar文件放在（D）下

A、应用程序的src目录

B、应用程序的/WEB-INF/lib目录

C、Tomcat的根目录

D、Tomcat的common/lib目录

7、INSERT 、UPDATE 、DELETE 语句是\_\_\_\_\_语句？( D )

A、DQL B、DCL C、DDL D、DML

8. 包含事务控制方法setAutoCommit,commit,rollback的接口是( A )

A.Connection

B.Statement

C.ResultSet

D.DriverManager

9. 下面的描述错误的是 (  B )

A、Statement的executeQuery()方法会返回一个结果集

B、Statement的executeUpdate()方法会返回是否更新成功的boolean值

C、使用ResultSet中的getString()可以获得一个对应于数据库中char类型的值

D、ResultSet中的next()方法会使结果集中的下一行成为当前行

10、建立数据库连接的方法是调用DriverManager类的（）静态方法

A.JDBC URL

B.setAutoCommit

C.Statement

D.getConnection

11、使用JdbcTemplate API操作数据库，（ C）方法能够插入一条数据。

A．query()

B．insert()

C．update()

D．save()

12、spring中哪个子框架能操作数据库（A）

A.template B.jdbc Cado.net D.以上都是

13、下列哪种不是JDBC支持的Statement对象？（B）

A.Statement

B.PrecallStatement

C.PreparedStatement

D.CallableStatement

14、以下哪一项不是SpringJDBC相关类型（ ）

A． JdbcTemplate

B． JdbcDAOSupport

C． JDBCUtils

D． DriverManagerDataSource

1. 多选
2. 要想使用Spring中的JDBC模块，就必须引入相应的jar文件（AB）

A．spring-jdbc

B．spring-tx

C．spring-web

D．spring-webmvc

2、update()方法的重载形式有（A B）【选两项】

A. int update(String sql)

B. int update(PreparedStatementCreatorpsc)

C. int update(PreparedStatementSetterpss , String sql)

D. int update(Object...args , String sql)

3、以下哪个是接口：（A B C ）

 A、ResultSet B、PreparedStatement  
 C、Statement D、DriverManager

4. 事务的4个特性？（）

A.原子性 B. 一致性 C. 隔离性 D. 持久性 E 传递性 F化学性

5、select empno no,ename name,sal salary from emp ,如下选项中可以获得的第一列数据的是（）

A. rs.getString(0)

B.rs.getString("empno")

C.rs.getString(1)

D.rs.getString("NO")

6、下列叙述不正确的是（ABC）

A.调用DriverManager 类和getConnection（）方法可以获得连接对象。

B.调用Connection对象的createStatement（）方法可以获得Statement

C.调用Statement对象的executeQuery()方法可以得到查询的结果集

D. 调用Connection对象和prepareStatement（）方法可以获得Statement对象

7、使用JdbcTemplate API进行查询操作数据库，下列说法正确的是（A、C ）。

A．查询表记录总数能用queryForInt()方法实现

B．查询表记录总数不能使用queryForObject()方法实现

C．queryForList()方法返回一个List，其中List里的每个对象一定是个Map

D．queryForList()方法返回一个List，其中List里的每个对象一定是个实体类对象

8、用于连接数据库的技术有？（A、B）

A.template B.jdbc Cado.net D.以上都是

9、 下列关于JDBC说法正确的是？（A、B）

A. JDBC是JavaEE中关于连接数据库的一系列接口和类规范的集合

B. 只要第三方厂商支持JDBC，则通过使用JDBC可以连接该数据库进行CRUD操作

C. 使用JDBC需要我们自己实现接口；

D. 连接不同数据库，JDBC驱动类可以使用同一类型;

E. 使用JDBC可以连接NoSQL数据库；

F. Class.forName方法可以获得连接；

10、 以下哪些为spring中支持的连接数据库的方式?（ ）

A． JDBC

B． ODBC

C． JNDI

D． 以上都不支持

1. 填空题
2. 我们可以使用 Spring 内置的类\_\_\_\_\_\_org.springframework.jdbc.datasource.SimpleDriverDataSource\_\_\_\_\_\_ 来配置数据库资源
3. Spring框架提供JDBC支持主要由4个包组成，分别是core（核心包）、object（对象包）、dataSource（数据源包）和support（支持包）
4. JDBC 全称：\_\_\_\_\_\_\_\_\_\_\_、\_\_\_\_\_\_\_\_、\_\_\_\_\_\_\_\_\_\_ 。（ Java Database Connectivity ）
5. 使用Class.forName()方法加载和注册驱动程序后，由\_\_DriverManager \_类负责管理并跟踪JDBC驱动程序，在数据库和相应驱动程序之间建立连接。
6. JDBC中的事务有哪些特点：\_原子性\_\_、\_一致性\_\_、\_\_隔离性\_、\_\_持久性\_
7. JDBC中使用PreparedStatement进行批量的数据操作时，可使用\_\_addBatch()\_\_和\_\_\_executeBatch()\_\_方法
8. 事务的4个特性\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_、\_\_\_\_\_\_\_\_\_、\_\_\_\_\_\_\_\_\_。()
9. spring 实现事务的2种方式：\_\_\_\_\_\_\_\_\_、\_\_\_\_\_\_\_\_\_（）
10. Spring JDBC抽象框架的四个组成部分是：\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_、\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_。（datasource、support、core、object）
11. 连接数据库导入哪个包\_\_\_\_\_\_\_\_\_\_
12. PreparedStatement执行增删改sql的方法名是 ，执行查询sql的方法名是 。(executeUpdate、executeQuery)
13. Spring中操作数据库的核心类为 。（JdbcTemplate）
14. JdbcTemplate中查询和更新数据的方法分别为： 和 。（query、update）
15. 判断题
16. Spring的数据访问层是以统一的数据访问异常层体系为核心，结合JDBC API的最佳实践和统一集成各种ORM方案，完成Java平台的数据访问（T）
17. Spring提供两种JDBC API的最佳实践，一种是以JDBCTemplate为核心的基于Template的JDBC使用方式，另一种则是在JdbcTemplate基础之上构建的基于操作对象的JDBC使用方式（T）
18. Spring除了提供基于Template形式的JDBC形式，还对各种数据库操作以面向对象的形式进行建模（T）
19. 使用<mvc:annotation-driven>启动验证框架。（ √ ）
20. JDBC中PreparedStatement使用冒号作为占位符(F)
21. Oracle的jdbc驱动类的包名和类名是oracle.jdbc.driver.OracleDriver（ T ）
22. spring jdbc不用处理事务（）
23. 从Spring 3.1开始，SimpleJdbcTemplate类已经被废弃了，在开发时不建议再使用。（ T ）
24. 一般我们使用jndi数据源的时候，服务器一般用的是weblogic和websphere等提供数据源的服务器。（T）
25. Template不能操作oracle数据库。（ ）
26. CallableStatement可以调用存储过程。（ T ）
27. JDBC默认事务需要手动提交。（F）
28. Clob/Blob是用来处理非二进制结果集数据。（T）
29. Spring中,我们可以选择DBCP,C3P0来配置数据源。（ T ）

五、简答题

# 1、Spring对当前各种流行的ORM解决方案的集成主要体现在几个方面?

答案：统一的资源管理方式

特定于ORM的数据访问异常到Spring统一异常体系的转译

统一的数据访问事务管理及控制方式

2、使用spring中的JDBC好处是什么？

答案：Spring中的JDBC为我们省去连接和关闭数据库的代码，我们着重关注对数据库的操作。Spring框架在数据库开发中的应用主要使用的是JDBCTemplate类，该类作为Spring对JDBC支持的核心，提供了所有对数据库操作功能的支持。我们将使用JDBCTemplate类完成对mysql数据库的增、删、改、查等操作。

3、简述jdbc查询数据库的基本流程

答案：JDBC调用数据库的基本步骤导入必要的类，装入JDBC驱动程序，识别数据源，分配一个Connection对象，分配一个Statement对象，使用Statement执行一个查询，从返回

的ResultSet对象中检索数据，关闭ResultSet，关闭Statement对象，关闭Connection对象。4、说出数据连接池的工作机制是什么?

Java服务器启动时会建立一定数量的池连接，并一直维持不少于此数目的池连接。客户端程序需要连接时，池驱动程序会返回一个未使用的池连接并将其表记为忙。如果当前没有空闲连接，池驱动程序就新建一定数量的连接，新建连接的数量有配置参数决定。当使用的池连接调用完成后，池驱动程序将此连接表记为空闲，其他调用就可以使用这个连接。

5、在JDBC中CLASS.FORNAME的作用 ?

答案：

1、使用反射技术检查数据类型是否存在

2、通过静态代码块给静态变量初始化——静态变量与JVM的方法区

2、 通过ClassLoader将数据类型加载到方法区

3、 在DriverManager中注册驱动

6、事务的4个特性：

原子性：一个事务中所有对数据库的操作是一个不可分割的操作序列，要么全做，要么全部做。  
一致性：数据不会因为事务的执行而遭到破坏。  
隔离性：一个事务的执行，不受其他事务（进程）的干扰。既并发执行的个事务之间互不干扰。  
持久性：一个事务一旦提交，它对数据库的改变将是永久的。

7、事务的实现方式？

实现方式共有两种：编码方式；声明式事务管理方式。  
基于AOP技术实现的声明式事务管理，实质就是：在方法执行前后进行拦截，然后在目标方法开始之前创建并加入事务，执行完目标方法后根据执行情况提交或回滚事务。  
声明式事务管理又有两种方式：基于XML配置文件的方式；另一个是在业务方法上进行@Transactional注解，将事务规则应用到业务逻辑中。

8、创建事务的时机？

是否需要创建事务，是由事务传播行为控制的。读数据不需要或只为其指定只读事务，而数据的插入，修改，删除就需要事务管理了。

9、根据代理机制的不同，列举五种Spring事务的配置方式一种？

第一种方式：每个Bean都有一个代理

1. **<bean** id="sessionFactory"
2. class="org.springframework.orm.hibernate3.LocalSessionFactoryBean"**>**
3. **<property** name="configLocation" value="classpath:hibernate.cfg.xml" **/>**
4. **<property** name="configurationClass" value="org.hibernate.cfg.AnnotationConfiguration" **/>**
5. **</bean>**
7. <!-- 定义事务管理器（声明式的事务） -->
8. **<bean** id="transactionManager"
9. class="org.springframework.orm.hibernate3.HibernateTransactionManager"**>**
10. **<property** name="sessionFactory" ref="sessionFactory" **/>**
11. **</bean>**
13. <!-- 配置DAO -->
14. **<bean** id="userDaoTarget" class="com.bluesky.spring.dao.UserDaoImpl"**>**
15. **<property** name="sessionFactory" ref="sessionFactory" **/>**
16. **</bean>**
18. **<bean** id="userDao"
19. class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean"**>**
20. <!-- 配置事务管理器 -->
21. **<property** name="transactionManager" ref="transactionManager" **/>**
22. **<property** name="target" ref="userDaoTarget" **/>**
23. **<property** name="proxyInterfaces" value="com.bluesky.spring.dao.GeneratorDao" **/>**
24. <!-- 配置事务属性 -->
25. **<property** name="transactionAttributes"**>**
26. **<props>**
27. **<prop** key="\*"**>**PROPAGATION\_REQUIRED**</prop>**
28. **</props>**
29. **</property>**
30. **</bean>**
31. **</beans>**

第二种方式：所有Bean共享一个代理基类

1. **<bean** id="sessionFactory"
2. class="org.springframework.orm.hibernate3.LocalSessionFactoryBean"**>**
3. **<property** name="configLocation" value="classpath:hibernate.cfg.xml" **/>**
4. **<property** name="configurationClass" value="org.hibernate.cfg.AnnotationConfiguration" **/>**
5. **</bean>**
7. <!-- 定义事务管理器（声明式的事务） -->
8. **<bean** id="transactionManager"
9. class="org.springframework.orm.hibernate3.HibernateTransactionManager"**>**
10. **<property** name="sessionFactory" ref="sessionFactory" **/>**
11. **</bean>**
13. **<bean** id="transactionBase"
14. class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean"
15. lazy-init="true" abstract="true"**>**
16. <!-- 配置事务管理器 -->
17. **<property** name="transactionManager" ref="transactionManager" **/>**
18. <!-- 配置事务属性 -->
19. **<property** name="transactionAttributes"**>**
20. **<props>**
21. **<prop** key="\*"**>**PROPAGATION\_REQUIRED**</prop>**
22. **</props>**
23. **</property>**
24. **</bean>**
26. <!-- 配置DAO -->
27. **<bean** id="userDaoTarget" class="com.bluesky.spring.dao.UserDaoImpl"**>**
28. **<property** name="sessionFactory" ref="sessionFactory" **/>**
29. **</bean>**
31. **<bean** id="userDao" parent="transactionBase" **>**
32. **<property** name="target" ref="userDaoTarget" **/>**
33. **</bean>**
34. **</beans>**

第三种方式：使用拦截器

1. **<bean** id="sessionFactory"
2. class="org.springframework.orm.hibernate3.LocalSessionFactoryBean"**>**
3. **<property** name="configLocation" value="classpath:hibernate.cfg.xml" **/>**
4. **<property** name="configurationClass" value="org.hibernate.cfg.AnnotationConfiguration" **/>**
5. **</bean>**
7. <!-- 定义事务管理器（声明式的事务） -->
8. **<bean** id="transactionManager"
9. class="org.springframework.orm.hibernate3.HibernateTransactionManager"**>**
10. **<property** name="sessionFactory" ref="sessionFactory" **/>**
11. **</bean>**
13. **<bean** id="transactionInterceptor"
14. class="org.springframework.transaction.interceptor.TransactionInterceptor"**>**
15. **<property** name="transactionManager" ref="transactionManager" **/>**
16. <!-- 配置事务属性 -->
17. **<property** name="transactionAttributes"**>**
18. **<props>**
19. **<prop** key="\*"**>**PROPAGATION\_REQUIRED**</prop>**
20. **</props>**
21. **</property>**
22. **</bean>**
24. **<bean** class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator"**>**
25. **<property** name="beanNames"**>**
26. **<list>**
27. **<value>**\*Dao**</value>**
28. **</list>**
29. **</property>**
30. **<property** name="interceptorNames"**>**
31. **<list>**
32. **<value>**transactionInterceptor**</value>**
33. **</list>**
34. **</property>**
35. **</bean>**
37. <!-- 配置DAO -->
38. **<bean** id="userDao" class="com.bluesky.spring.dao.UserDaoImpl"**>**
39. **<property** name="sessionFactory" ref="sessionFactory" **/>**
40. **</bean>**
41. **</beans>**

第四种方式：使用tx标签配置的拦截器

1. **<bean** id="sessionFactory"
2. class="org.springframework.orm.hibernate3.LocalSessionFactoryBean"**>**
3. **<property** name="configLocation" value="classpath:hibernate.cfg.xml" **/>**
4. **<property** name="configurationClass" value="org.hibernate.cfg.AnnotationConfiguration" **/>**
5. **</bean>**
7. <!-- 定义事务管理器（声明式的事务） -->
8. **<bean** id="transactionManager"
9. class="org.springframework.orm.hibernate3.HibernateTransactionManager"**>**
10. **<property** name="sessionFactory" ref="sessionFactory" **/>**
11. **</bean>**
13. **<tx:advice** id="txAdvice" transaction-manager="transactionManager"**>**
14. **<tx:attributes>**
15. **<tx:method** name="\*" propagation="REQUIRED" **/>**
16. **</tx:attributes>**
17. **</tx:advice>**
19. **<aop:config>**
20. **<aop:pointcut** id="interceptorPointCuts"
21. expression="execution(\* com.bluesky.spring.dao.\*.\*(..))" **/>**
22. **<aop:advisor** advice-ref="txAdvice"
23. pointcut-ref="interceptorPointCuts" **/>**
24. **</aop:config>**
25. **</beans>**

第五种方式：全注

2. **<tx:annotation-driven** transaction-manager="transactionManager"**/>**
4. **<bean** id="sessionFactory"
5. class="org.springframework.orm.hibernate3.LocalSessionFactoryBean"**>**
6. **<property** name="configLocation" value="classpath:hibernate.cfg.xml" **/>**
7. **<property** name="configurationClass" value="org.hibernate.cfg.AnnotationConfiguration" **/>**
8. **</bean>**
10. <!-- 定义事务管理器（声明式的事务） -->
11. **<bean** id="transactionManager"
12. class="org.springframework.orm.hibernate3.HibernateTransactionManager"**>**
13. **<property** name="sessionFactory" ref="sessionFactory" **/>**
14. **</bean>**
16. **</beans>**

10、spring基于xml配置声明式事务管理步骤

步骤:

1.在applicationContext.xml中声明aop和tx的名称空间

2.声明事物管理器  也就是声明一个bean

3.声明通知  通过<tx:advice>和他里面的各种属性配置其中name是必须的,说明对那个方法进行事物控制

4.声明切面  通过<aop:config>配置切面,在里面配置切点和通知

### 11、在Spring框架中如何更有效地使用JDBC

使用SpringJDBC 框架，资源管理和错误处理的代价都会被减轻。所以开发者只需写statements 和 queries从数据存取数据，JDBC也可以在Spring框架提供的模板类的帮助下更有效地被使用，这个模板叫JdbcTemplate 。

12、Spring通过什么设计模式对JDBC操作进行支持？

答案：

Spring通过模版模式对JDBC操作进行了封装支持。模板模式：将程序中的可变的和不可变的部分分离，定义操作的步骤（固定的），将可变的步骤交给回调处理，回调（命令）：由组件定义，但不由组件调用，而是由系统调用。

13、Spring JDBC和普通JDBC操作有什么优势？

答案：

Spring提供了一个工具类JdbcTemple，该类对jdbc的操作进行了轻量级别的封装。

优点如下：

大大简化了代码开发难度，开发人员无需关注固定的流程性代码，而只关注查询语句和结果处理；

直接使用sql语句操作数据库，效率很高；

支持数据库的分区，这是数据量大的项目的实现方案，hibernate无法实现；

使用java语言拼sql语句，效率很高，这是Mybatis无法达到的。

14、JdbcTemplate有什么功能【简答题】【简单】

答案：

JdbcTemplate类通过模板设计模式帮助消除冗长的代码，只做需要做的事情（即可变部分），并且自动完成固定部分，如连接的创建及关闭，JdbcTemplate类对可变部分采用回调接口方式实现，如ConnectionCallback通过回调接口返回给用户一个连接，从而可以使用该连接做任何事情、StatementCallback通过回调接口返回给用户一个Statement，从而可以使用该Statement做任何事情等等。

15、事务有什么特点？

答案：

事务首先是一系列操作组成的工作单元，该工作单元内的操作是不可分割的，即要么所有操作都做，要么所有操作都不做，这就是事务。

事务必需满足ACID（原子性、一致性、隔离性和持久性）特性，缺一不可：

原子性（Atomicity）：

即事务是不可分割的最小工作单元，事务内的操作要么全做，要么全不做

一致性（Consistency）：

在事务执行前数据库的数据处于正确的状态，而事务执行完成后数据库的数据还是处于正确的状态，即数据完整性约束没有被破坏；如银行转帐，A转帐给B，必须保证A的钱一定转给B，一定不会出现A的钱转了但B没收到，否则数据库的数据就处于不一致（不正确）的状态

隔离性（Isolation）：

并发事务执行之间无影响，在一个事务内部的操作对其他事务是不产生影响，这需要事务隔离级别来指定隔离性

持久性（Durability）：

事务一旦执行成功，它对数据库的数据的改变必须是永久的，不会因比如遇到系统故障或断电造成数据不一致或丢失。

16、什么是事务的隔离级别？

答案：

在实际开发中数据库操作一般都是并发执行的，即有多个事务并发执行，并发执行就可能遇到问题，目前常见的问题如下：

丢失更新：

两个事务同时更新一行数据，最后一个事务的更新会覆盖掉第一个事务的更新，从而导致第一个事务更新的数据丢失，这是由于没有加锁造成的

脏读：

一个事务看到了另一个事务未提交的更新数据

不可重复读：

在同一事务中，多次读取同一数据却返回不同的结果；也就是有其他事务更改了这些数据

幻读：

一个事务在执行过程中读取到了另一个事务已提交的插入数据；即在第一个事务开始时读取到一批数据，但此后另一个事务又插入了新数据并提交，此时第一个事务又读取这批数据但发现多了一条，即好像发生幻觉一样

在标准SQL规范中通过定义四种数据库隔离级别来解决并发：

隔离级别 含义

DEFAULT 使用后端数据库默认的隔离级别(spring中的的选择项)

READ\_UNCOMMITED 允许你读取还未提交的改变了的数据。可能导致脏、幻、不可重复读

READ\_COMMITTED 允许在并发事务已经提交后读取。可防止脏读，但幻读和 不可重复读仍可发生

REPEATABLE\_READ 对相同字段的多次读取是一致的，除非数据被事务本身改变。可防止脏、不可重复读，但幻读仍可能发生。

SERIALIZABLE 完全服从ACID的隔离级别，确保不发生脏、幻、不可重复读。这在所有的隔离级别中是最慢的，它是典型的通过完全锁定在事务中涉及的数据表来完成的。

隔离级别越高，数据库事务并发执行性能越差，能处理的操作越少。因此在实际项目开发中为了考虑并发性能一般使用提交读隔离级别，它能避免丢失更新和脏读，尽管不可重复读和幻读不能避免，可以在可能出现的场合使用悲观锁或乐观锁来解决这些问题。

17、什么是事务的传播特性？

答案：

spring控制事务是以bean组件的方法为单位的，如果一个方法正常执行完毕，该方法内的全部数据库操作按照一次事务提交，如果抛出异常，全部回滚

如两个bean组件都由spring控制事务，且组件的方法之间存在调用关系，即（bean1 方法a 调用了 bean2 方法b），spring提供了一组配置方式供开发者选择，这些配置方式称为事务的传播策略

事务的隔离级别是数据库本身的事务功能，然而事务的传播属性则是Spring自己为我们提供的功能，数据库事务没有事务的传播属性这一说法

传播行为 意义

REQUIRED 业务方法需要在一个事务中运行。如果方法运行时，已经处在一个事务中，那么加入到该事务，否则为自己创建一个新的事务

NOT\_SUPPORTED 声明方法不需要事务。如果方法没有关联到一个事务，容器不会为它开启事务。如果方法在一个事务中被调用，该事务会被挂起，在方法调用结束后，原先的事务便会恢复执行

REQUIRESNEW 属性表明不管是否存在事务，业务方法总会为自己发起一个新的事务。如果方法已经运行在一个事务中，则原有事务会被挂起，新的事务会被创建，直到方法执行结束，新事务才算结束，原先的事务才会恢复执行

MANDATORY 该属性指定业务方法只能在一个已经存在的事务中执行，业务方法不能发起自己的事务。如果业务方法在没有事务的环境下调用，容器就会抛出例外

SUPPORTS 这一事务属性表明，如果业务方法在某个事务范围内被调用，则方法成为该事务的一部分。如果业务方法在事务范围外被调用，则方法在没有事务的环境下执行

NEVER 指定业务方法绝对不能在事务范围内执行。如果业务方法在某个事务中执行，容器会抛出例外，只有业务方法没有关联到任何事务，才能正常执行

NESTED 如果一个活动的事务存在，则运行在一个嵌套的事务中.，如果没有活动事务, 则按REQUIRED属性执行.它使用了一个单独的事务， 这个事务拥有多个可以回滚的保存点。内部事务的回滚不会对外部事务造成影响。它只对DataSourceTransactionManager事务管理器起效

18、Spring对事务支持的特点是什么？

答案：

Spring框架最核心功能之一就是事务管理，而且提供一致的事务管理抽象，这能帮助我们：

提供一致的编程式事务管理API，不管使用Spring JDBC框架还是集成第三方框架使用该API进行事务编程。

无侵入式的声明式事务支持；

Spring支持声明式事务和编程式事务事务类型；

在Spring中，只需要在Spring配置文件中做一些配置，即可将数据库的访问纳入到事务管理中，解除了和代码的耦合， 这是对应用代码影响最小的选择。当不需要事务管理的时候，可以直接从Spring配置文件中移除该设置。

19、编程式事务管理和声明式事务管理有什么区别？

答案：

编程式事务需要你在代码中直接加入处理事务的逻辑,可能需要在代码中显式调用commit()、rollback()等事务管理相关的方法,如在执行a方法时候需要事务处理,你需要在a方法开始时候开启事务,处理完后，在方法结束时候,关闭事务。声明式的事务的做法是在a方法外围添加注解或者直接在配置文件中定义,a方法需要事务处理,在spring中会通过配置文件在a方法前后拦截,并添加事务。二者区别：编程式事务侵入性比较强，但处理粒度更细。

20、jdbc操作数据库时，由哪些常用jdbc类，写出三个既可

答案：

21、如何实现防sql注入式攻击

22、简述使用JDBC操作数据库的主要步骤。

答：

（1） 加载驱动类；

（2） 获得连接对象；

（3） 获得语句对象；

（4） 执行SQL语句，如果是查询语句，需要处理结果集；

（5） 关闭结果集。语句对象。数据库连接对象。

23、简述Spring对Jdbc的支持？

答案：

Spring JDBC 是spring 官方提供的一个持久层框架，对jdbc进行了抽象和封装，消除了重复冗余的jdbc重复性的代码，使操作数据库变的更简单。

但Spring JDBC本身并不是一个orm框架，与hibernate相比，它需要自己操作sql,手动映射字段关系，在保持灵活性的同时，势必造成了开发效率的降低。

24、JNDI和JDBC的区别和联系及其使用方法

两者都是连接数据库的标准

JDBC

全称：Java Database Connectivity

以一种统一的方式来对各种各样的数据库进行存取

使用JDBC带来的问题：

(1).数据库服务器名称 、用户名和口令都可能需要改变，由此引发JDBC URL需要修改

(2).数据库可能改用别的产品，如改用DB2或者Oracle，引发JDBC驱动程序包和类名需要修改

(3).随着实际使用终端的增加，原配置的连接池参数可能需要调整

JNDI

Java Naming and Directory Interface

它提供了一致的模型来存取和操作企业级的资源如DNS和LDAP

JNDI相对与JDBC来说是他的灵活性,程序员不需要关心“具体的数据库后台是什么？JDBC驱动程序是什么？JDBC URL格式是什么？访问数据库的用户名和口令是什么？”等等这些问题，程序员编写的程序应该没有对 JDBC 驱动程序的引用，没有服务器名称，没有用户名称或口令 —— 甚至没有数据库池或连接管理。而是把这些问题交给J2EE容器来配置和管理，程序员只需要对这些配置和管理进行引用即可。

25、Spring通过什么方式访问Hibernate

在Spring中有两种方式访问Hibernate：

• 控制反转 Hibernate Template和 Callback。

• 继承 HibernateDAOSupport提供一个AOP 拦截器。

26、列出3中spring支持的ORM

• Hibernate

• iBatis

• JPA (Java Persistence API)

• TopLink

• JDO (Java Data Objects)

六、程序实现题

1、有一张student的表字段为id,name,已id为条件通过Spring对JDBC的操作取出对应的name

|  |
| --- |
| 配置文件如applicationContext.xml主要代码：  <bean id="dateSource" class="org.springframework.jdbc.datasource.SimpleDriverDataSource"> <property name="username" value="root"/> <property name="password" value="root"/> <property name="driverClass" value="com.mysql.jdbc.Driver"/> <property name="url" value="jdbc://mysql://locolhost:3306/student"/> </bean>  JDBC 操作类：  import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Component; import pojo.Student; import javax.sql.DataSource; import java.sql.\*; @Component("jdbc") public class JDBCtest { @Autowired private DataSource dataSource; public Student getOne(int stuID) throws SQLException { String sql = "SELECT id, name FROM student WHERE id = " + stuID; Student student = new Student(); Connection con = dataSource.getConnection(); Statement st = con.createStatement(); ResultSet rs = st.executeQuery(sql); if (rs.next()) { student.setId(rs.getInt("id")); student.setName(rs.getString("name")); } return student; } }  测试类：  ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml"); JDBCtest jdbc = (JDBCtest) context.getBean("jdbc"); Student student = jdbc.getOne(123456789); System.out.println(student.getId()); System.out.println(student.getName()); |

2、请编写applicationContext.xml文件中JDBC的配置信息

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"  "http://www.springframework.org/dtd/spring-beans.dtd">  <beans>  <!-- 配置数据源 -->  <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">  <!-- 数据库驱动 -->  <property name="driverClassName" value="com.mysql.jdbc.Driver"/>  <!-- 连接数据库的URL 数据库名为已经创建好的User -->  <property name="url" value="jdbc:mysql://localhost/User"/>  <!-- 连接数据库的用户名 -->  <property name="username" value="root"/>  <!-- 连接数据的密码 -->  <property name="password" value="root"/>  </bean>  <!-- 配置JDBC模板 -->  <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">  <!-- 默认必须使用数据源 -->  <property name="dataSource" ref="dataSource"/>  </bean>  </beans> |

3、使用jdbc完成以下操作说明：

1），注册驱动;(只需要注册一次)

2），建立连接；

3），创建执行的sql语句；

4），执行语句；

5），处理执行结果；

6），释放资源；

（10分）

|  |
| --- |
| import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.ResultSet;  import com.mysql.jdbc.Statement;  public class Demo1 {  public static void main(String[] args) throws Exception {  test();  }  public static void test() throws Exception {  //1,注册驱动  Class.forName("com.mysql.jdbc.Driver")  //2，建立连接  String url = "jdbc:mysql://localhost:3306/test";  String user = "root";  String password = "123456";  Connection conn = (Connection)DriverManager.getConnection(url, user, password);  //3，创建slq执行语句  Statement st = (Statement) conn.createStatement();  //4,执行语句  ResultSet rs = st.executeQuery("select \* from a");  //5，处理执行结果  while (rs.next()) {  System.out.println(rs.getObject(1));  }  //6，关闭连接，释放资源  rs.close();  st.close();  conn.close();  }  } |

4、用spring jdbc完成登录（10分）

5、用spring jdbc完成dept模块（10分）

6、用spring jdbc完成emp模块（10分）

7、如何通过[Spring配置JNDI和通过JNDI获取DataSource](https://www.cnblogs.com/houhaihu/p/6135330.html)？

|  |
| --- |
| 1、SpringJNDI数据源配置信息 <bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">  <property name="jndiName">  <value>java:comp/env/myDataSourceJNDI</value>  </property>  </bean>  上面<value>中myDataSourceJNDI是tomcat或者其他应用服务器配置的JNDI。  2、JNDI的配置(tomcat中)  修改tomcat目录conf/context.xml文件：  <Resource name="myDataSourceJNDI"  　　auth="Container" type="javax.sql.DataSource"  　　maxActive="100" maxIdle="30" maxWait="10000" username="root"  　　password="root" driverClassName="oracle.jdbc.driver.OracleDriver"  　　url="jdbc:oracle:thin:@127.0.0.1:1521:TEST"/>  3、通过JNDI获取DataSource  Context context = new InitialContext();  DataSource ds = (DataSource)context.lookup("java:comp/env/myDataSourceJNDI"); |

8、针对Spring的JdbcTemplate的query方法实现一个通用的自动利用结果集列表名和实体类属性名进行匹配的RowMapper。

public class ParameterNameRowMapper<T> implements RowMapper<T> {

private Class<T> persistenceClass;

private BeanUtil beanUtil = new BeanUtil();

public ParameterNameRowMapper(Class<T> clz) {

persistenceClass = clz;

}

@Override

public T mapRow(ResultSet rs, int rowNum) throws SQLException {

// TODO Auto-generated method stub

return resultSet2List(rs);

}

/\*\*

\* 将JDBC结果集转化为实体对象集合的方法

\*

\* @param rs

\* 从数据库中查询得到的结果集对象

\* @param persistenceClass

\* 对应的实体类信息

\* @return 最终转化的结果集合

\*/

private T resultSet2List(ResultSet rs) {

// 创建结果集合

// ArrayList<T> resultList = new ArrayList<T>();

try {

// 获取结果集的元数据，用以获取结果集中的总列数，每个列的列名等

ResultSetMetaData rsMetaData = rs.getMetaData();

// 得到本结果集中的总列数

int columnCount = rsMetaData.getColumnCount();

// 循环遍历结果集

// while (rs.next()) {

// 创建单一的实体对象

T entityObject = persistenceClass.newInstance();

// 循环遍历本行的所有单元格

for (int cindex = 1; cindex <= columnCount; cindex++) {

// 获取到当前单元的对应的列名

String cname = rsMetaData.getColumnName(cindex);

// 判定单元格中数据的JDBC类型

switch (rsMetaData.getColumnType(cindex)) {

// 处理BLOB类型

case BLOB:

// 获取BLOB类型数据

Blob blob = rs.getBlob(cindex);

// 填充实体对象中的相关属性

beanUtil.setBeanProperty(entityObject, cname, blob, true);

break;

// 处理CLOB类型

case CLOB:

// 获取CLOB类型数据

Clob clob = rs.getClob(cindex);

// 填充实体对象中的相关属性

beanUtil.setBeanProperty(entityObject, cname, clob, true);

break;

// 处理日期时间类型

case TIME:

case TIMESTAMP:

case DATE:

// 获取日期数据

// System.out.println(rs.getTimestamp(cindex));

Timestamp time = rs.getTimestamp(cindex);

if (time != null) {

// long timestamp=rs.getTimestamp(cindex).getTime();

Date date = new Date(time.getTime());

// 填充实体对象中的相关属性

beanUtil.setBeanProperty(entityObject, cname, date, true);

}

break;

// 处理其他日常数据

default:

// 以字符串的形式获取数据值

String value = rs.getString(cindex);

// 利用转换器进行格式转换后填充相关的属性

//System.out.println(cname+" "+value);

beanUtil.setBeanProperty(entityObject, cname, value, true);

break;

}

}

// 将本次创建的实体对象加入结果集合

// resultList.add(entityObject);

return entityObject;

// }

} catch (Exception e) {

// 如果执行的过程中遇到异常，则输出异常信息

e.printStackTrace();

// 在有异常的情况下，返回null

return null;

}

// 返回最终的结果集合

// return resultList;

}

}

9、使用JDBC查询数据库中名为t\_userinfo表的数据

|  |
| --- |
| Class.forName(“”);  Connection con = DriverManager.getConnection(“ ”,””,””);  String sql = “select \* from t\_userinfo”;  PreparedStatement pstm = con.prepareStatement(sql);  ResultSet rs = pstm.executeQuery();  While(rs.next()){  String userName = rs.getString(“t\_username”);  //…  }  rs.close();  pstm.close();  con.close(); |

10、书写出spring通过JDBC连接mysql数据库的配置，只需列出配置文件即可

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN" "http://www.springframework.org/dtd/spring-beans-2.0.dtd">  <beans>  <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">  <property name="driverClassName">  <value>com.mysql.jdbc.Driver</value>  </property>  <property name="url">  <value>jdbc:mysql://localhost:3306/test</value>  </property>  <property name="username">  <value>root</value>  </property>  <property name="password">  <value>xxxxxx</value>  </property>  </bean>  <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">  <property name="dataSource">  <ref local="dataSource"/>  </property>  </bean>  <bean id="userDAO" class="springjdbc.dao.impl.UserDaoImpl">  <property name="jdbcTemplate">  <ref bean="jdbcTemplate"/>  </property>  </bean>  <bean id="user" class="springjdbc.pojo.User">  <property name="dao">  <ref bean="userDAO"/>  </property>  </bean>  </beans> |