# 事务

事务：事务指逻辑上的一组操作，组成这组操作的各个单元，要么全部成功，要么全部不成功。

例如：A——B转帐，对应于如下两条sql语句

update account set money=money-100 where name=‘A’;

update account set money=money+100 where name=‘B’;

## Mysql中的事务

* mysql引擎是支持事务的
* mysql默认自动提交事务。每条语句都处在单独的事务中。
* 手动控制事务
* 开启事务：start | begin transaction
* 提交事务：commit
* 回滚事务：rollback

## Oracle中的事务

Oracle不需要特别的去指定事务的开始和结束。一个事务的结束就是下一个事务的开始

开启事物：

1、连接到数据库，并执行第一条DML语句（insert delete update）

2、前一个事务结束后，又输入了另一条DML语句

提交事务：commit

回滚事务：rollback

## JDBC如何控制事务

当Jdbc程序向数据库获得一个Connection对象时，默认情况下这个Connection对象会自动向数据库提交在它上面发送的SQL语句。若想关闭这种默认提交方式，让多条SQL在一个事务中执行，可使用下列语句：

JDBC控制事务语句

Connection.setAutoCommit(false); // 开启事物 手动管理事务

Connection.rollback(); // rollback 回滚

Connection.commit(); // commit 提交事务

|  |
| --- |
| **package** com.imcode;  **import** java.sql.Connection; **import** java.sql.DriverManager; **import** java.sql.SQLException; **import** java.sql.Statement;  /\*\*  \* 事务控制  \* 手动控制事务  \*  \* 在需要事务控制的SQL执行之前调用 conn.setAutoCommit(false);  \* SQL语句如果全部执行正确 提交事务 conn.commit();  \* 如果出现异常 回滚事务 conn.rollback();  \* 最后释放资源 conn.close(); 如果有未提交的事务，提交事务，关闭数据库连接  \*/ **public class** JDBCDemo04 {   **public static void** main(String[] args) {  Connection conn = **null**;  Statement st = **null**;  **try** {  String url = "jdbc:oracle:thin:@127.0.0.1:1521:orcl";  String username = "app";  String password = "123456";  conn = DriverManager.*getConnection*(url, username, password);  // 关闭事务的自动提交 手动控制事务  conn.setAutoCommit(**false**);   st = conn.createStatement();  // 执行转账的动作  String sql1 = "UPDATE T\_ACCOUNT SET MONEY = MONEY - 500 WHERE ACCOUNT='JACK'";  String sql2 = "UPDATE T\_ACCOUNT SET MONEY = MONEY + 500 WHERE ACCOUNT='ROSE'";   st.executeUpdate(sql1);  st.executeUpdate(sql2);  System.***out***.println("转账已经成功 已经成功 已经成功");  // 手动提交事务  conn.commit();   System.***out***.println("给两个账户发送成功消息通知");  **int** i = 100 / 0;  } **catch** (Exception e) {  // 如果出现异常回滚事务  **try** {  conn.rollback();  } **catch** (SQLException e1) {  e1.printStackTrace();  }  e.printStackTrace();  } **finally** {  **try** {  st.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  **try** {  conn.close();// 关闭连接的时候事务自动提交  } **catch** (SQLException e) {  e.printStackTrace();  }  }  } } |

## 事务的特性

* **原子性**

指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。

* **隔离性**

事务的隔离性是多个用户并发访问数据库时，数据库为每一个用户开启的事务，不能被其他事务的操作数据所干扰，多个并发事务之间要相互隔离。

* **持久性**

指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来即使数据库发生故障也不应该对其有任何影响。

* **一致性**

事务必须使数据库从一个一致性状态变换到另外一个一致性状态。转账前和转账后的总金额不变。

## 事务的隔离级别

概念

一个事务对数据库的修改与并行的另一个事务的隔离程度。

两个并发事务同时访问数据库表相同的行时，可能存在以下三个问题：

* **幻读：（insert）**

事务T1读取一条指定where条件的语句，返回结果集。此时事务T2插入一行新记录，恰好满足T1的where条件。

然后T1使用相同的条件再次查询，结果集中可以看到T2插入的记录，这条新纪录就是幻读。

* **不可重复读：（update）**

事务T1读取一行记录，紧接着事务T2修改了T1刚刚读取的记录，然后T1再次查询，发现与第一次读取的记录不同，这称为不可重复读。

* **脏读：**

事务T1更新了一行记录，还未提交所做的修改（没有提交事物），这个时候T2读取了更新后的数据，然后T1执行回滚操作，取消刚才的修改，

所以T2所读取的行就无效，也就是脏数据。

为了处理这些问题，SQL标准定义了以下几种事务隔离级别

READ UNCOMMITTED 幻读、不可重复读和脏读都允许。-- 读未提交

READ COMMITTED 允许幻读、不可重复读，不允许脏读（oracle默认的）-- 读已提交避免了脏读

REPEATABLE READ 允许幻读，不允许不可重复读，不允许脏读（mysql默认）--可重复读

SERIALIZABLE 幻读、不可重复读和脏读都不允许

级别越高，性能越低，数据越安全

Oracle数据库支持READ COMMITTED 和 SERIALIZABLE这两种事务隔离级别。所以Oracle不支持脏读



|  |  |  |  |
| --- | --- | --- | --- |
| 时间 | 线程1 事物1 | 线程2 事物2 | 说明 |
| t1 | begin; |  |  |
| t2 | select \* from account where name='A';  结果200块 |  |  |
| t3 |  | begin; |  |
| t4 |  | update account set money=money+100 where name='A'; |  |
| t5 | select \* from account where name='A';  结果300块 |  | 读到了另一个线程未提交事务的数据。**赃读**发生了 |
| t6 |  | commit; |  |
| t7 | select \* from account where name='A';  结果300块 |  | 读到了另一个线程提交事务的update数据。**不可重复读**发生了 |
| t8 |  | insert into account values('C',1000);  执行insert语句插入数据，事务自动提交了 |  |
| t9 | select \* from account;  查到4条数据 |  | 读到了另一个线程自动提交事务的insert语句数据。**幻读**发生了 |
| t10 | commit; |  |  |

## JDBC控制事务的隔离级别

**MySQL的事物隔离级别：**

show variables like '%isolation%';

select @@global.tx\_isolation, @@tx\_isolation;

修改隔离级别：

mysql> help ISOLATION

Name: 'ISOLATION'

Description:

Syntax:

SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL

{

REPEATABLE READ

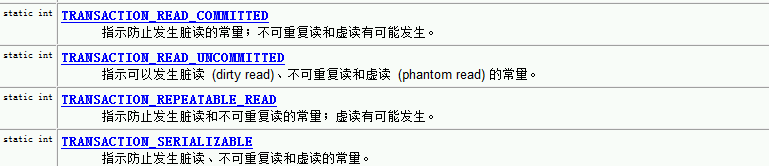
| READ COMMITTED

| READ UNCOMMITTED

| SERIALIZABLE

}

Connection接口：



设置隔离级别：必须在开启事务之前。

Connection.setTransactionIsolation(int level);

**Oracle设置事务隔离级别：**

oracle数据库支持READ COMMITTED 和 SERIALIZABLE这两种事务隔离级别。

默认系统事务隔离级别是READ COMMITTED,也就是读已提交

l         设置隔离级别  
  
l          设置一个事务的隔离级别  
  
l         SET TRANSACTION ISOLATION LEVEL READ COMMITTED;  
  
l         SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
  
l         SET TRANSACTION READ ONLY;  
  
l         设置增个会话的隔离级别  
  
l         ALTER SESSION SET ISOLATION\_LEVEL SERIALIZABLE;  
  
l         ALTER SESSION SET ISOLATION\_LEVEL READ COMMITTED;

查看session的会话隔离级别就是v$transaction 里查看，沒有任何事務的情況下這個視圖是沒有任何東西的..

# 数据库连接池

数据库连接池负责分配、管理和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接，而不是再重新建立一个；

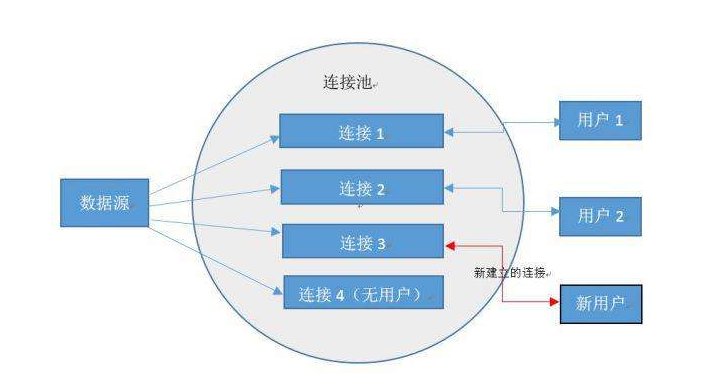
释放空闲时间超过最大空闲时间的数据库连接来避免因为没有释放数据库连接而引起的数据库连接遗漏。这项技术能明显提高对数据库操作的性能。

查询数据库当前设置的最大连接数：

show variables like '%max\_connections%';

## 连接池原理

目的：解决建立数据库连接耗费资源和时间很多的问题，提高性能。



## 读取配置文件

通过当前类加载器的getResourceAsStream方法获取

InputStream in=Test.**class**.getClassLoader().getResourceAsStream("test.properties");

从文件获取

InputStream inStream = **new** FileInputStream(**new** File("filePath"));

Dbcp druid

## DBCP

DBCP：Apache推出的Database Connection Pool

使用步骤：

> 添加jar包

commons-dbcp-1.4.jar

commons-pool-1.5.6.jar

commons-logging-1.5.6.jar

> 添加属性资源文件

> 编写数据源工具类

|  |
| --- |
| public class DBUtil {  private static DataSource ds = null;  static {  try{  InputStream in = DBUtil.class.getClassLoader()  .getResourceAsStream("ds.properties");  Properties props = new Properties();  props.load(in);  ds = BasicDataSourceFactory.createDataSource(props);  }catch(Exception ex){  ex.printStackTrace();  }  }    public static Connection getConnection() throws SQLException{  return ds.getConnection();  }  } |

Oracle 数据库连接数相关SQL

select count(\*) from v$process --当前的连接数

select value from v$parameter where name = 'processes' --数据库允许的最大连接数

## Druid

阿里巴巴推出的国产数据库连接池，据网上测试对比，比目前的DBCP或C3P0数据库连接池性能更好，Druid与其他数据库连接池使用方法基本一样（与DBCP非常相似），将数据库的连接信息全部配置给DataSource对象

|  |
| --- |
| public class DBUtil {  private static DataSource ds = null;  static {  try{  InputStream in = DBUtil.class.getClassLoader()  .getResourceAsStream("ds.properties");  Properties props = new Properties();  props.load(in);  ds = DruidDataSourceFactory.createDataSource(props);  }catch(Exception ex){  ex.printStackTrace();  }  }    public static Connection getConnection() throws SQLException{  return ds.getConnection();  }  } |

## DBCP配置参考

#连接设置

driverClassName=com.mysql.jdbc.Driver

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

username=root

password=123456

#初始化连接数量

initialSize=10

#最大连接数量

-------maxActive=50

#最大空闲连接

maxIdle=20

#最小空闲连接

minIdle=5

#超时等待时间以毫秒为单位 6000毫秒/1000等于60秒

maxWait=60000

#JDBC驱动建立连接时附带的连接属性属性的格式必须为这样：[属性名=property;]

#注意："user" 与 "password" 两个属性会被明确地传递，因此这里不需要包含他们。

connectionProperties=useUnicode=true;characterEncoding=utf-8

#指定由连接池所创建的连接的自动提交（auto-commit）状态。

defaultAutoCommit=true

#driver default 指定由连接池所创建的连接的只读（read-only）状态。

#如果没有设置该值，则“setReadOnly”方法将不被调用。（某些驱动并不支持只读模式，如：Informix）

defaultReadOnly=

#driver default 指定由连接池所创建的连接的事务级别（TransactionIsolation）。

#可用值为下列之一：（详情可见javadoc。）NONE,READ\_UNCOMMITTED, READ\_COMMITTED, REPEATABLE\_READ, SERIALIZABLE

defaultTransactionIsolation=READ\_UNCOMMITTED

## Druid配置参考

#数据库基本信息配置

driverClassName = oracle.jdbc.driver.OracleDriver

url = jdbc:oracle:thin:@127.0.0.1:1521:ORCL

username = \*\*\*\*\*\*

password = \*\*\*\*\*\*

#初始化连接数量

initialSize = 5

#最大并发连接数

maxActive = 10

#最大空闲连接数

maxIdle

#最小空闲连接数

minIdle = 3

#配置获取连接等待超时的时间

maxWait = 60000

#超过时间限制是否回收

removeAbandoned = true

#超过时间限制多长回收

removeAbandonedTimeout = 180

#配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒

timeBetweenEvictionRunsMillis = 60000

#配置一个连接在池中最小生存的时间，单位是毫秒

minEvictableIdleTimeMillis = 300000

#用来检测连接是否有效的sql，要求是一个查询语句

validationQuery = SELECT 1 FROM DUAL

#申请连接的时候检测

testWhileIdle = true

#申请连接时执行validationQuery检测连接是否有效，配置为true会降低性能

testOnBorrow = false

#归还连接时执行validationQuery检测连接是否有效，配置为true会降低性能

testOnReturn = false

#打开PSCache，并且指定每个连接上PSCache的大小

poolPreparedStatements = true

maxPoolPreparedStatementPerConnectionSize = 50

属性类型是字符串，通过别名的方式配置扩展插件，常用的插件有：

监控统计用的filter:stat

日志用的filter:log4j

防御SQL注入的filter:wall

filters = stat



