JDBC-2

讲师：宋红康
新浪微博：尚硅谷-宋红康

主要内容

1. JDBC概述
2. 获取数据库连接
3. 使用Statement操作数据表的弊端
4. 使用PreparedStatement
   - 实现数据表的INSERT/UPDATE/DELETE操作
   - 使用ResultSet和ResultSetMetaData实现数据表的SELECT操作
   - 向数据表中插入、读取大数据：BLOB字段
   - 使用PreparedStatement实现批量插入

主要内容

5. 数据库事务
6. 数据库连接池
   - C3P0数据库连接池
   - DBCP数据库连接池
7. DBUtils工具类
   - 使用QueryRunner，实现UPDATE()和QUERY()方法
   - 利用DbUtils编写DAO通用类

5-数据库事务

数据库事务

- **事务：**一组逻辑操作单元,使数据从一种状态变换到另一种状态。
- **事务处理（事务操作）：**保证所有事务都作为一个工作单元来执行，即使出现了故障，都不能改变这种执行方式。当在一个事务中执行多个操作时，要么所有的事务都被提交(commit)，那么这些修改就永久地保存下来；要么数据库管理系统将放弃所作的所有修改，整个事务回滚(rollback)到最初状态。
- 为确保数据库中数据的一致性,数据的操纵应当是离散的成组的逻辑单元:当它全部完成时,数据的一致性可以保持,而当这个单元中的一部分操作失败,整个事务应全部视为错误,所有从起始点以后的操作应全部回退到开始状态。

JDBC 事务处理

- 当一个连接对象被创建时，默认情况下是自动提交事务：每次执行一个 SQL 语句时，如果执行成功，就会向数据库自动提交，而不能回滚
- 为了让多个 SQL 语句作为一个事务执行：
  - 调用 Connection 对象的 setAutoCommit(false); 以取消自动提交事务
  - 在所有的 SQL 语句都成功执行后，调用 commit(); 方法提交事务
  - 在出现异常时，调用 rollback(); 方法回滚事务
  - 若此时 Connection 没有被关闭, 则需要恢复其自动提交状态

数据库事务使用的过程

```
public void testJDBCTransaction() {
Connection conn = null;
try {
// 1.获取数据库连接
conn = JDBCUtils.getConnection();
// 2.开启事务
conn.setAutoCommit(false);
// 3.进行数据库操作

// 4.若没有异常，则提交事务
conn.commit();
} catch (Exception e) {
e.printStackTrace();
// 5.若有异常，则回滚事务
try {
conn.rollback();
} catch (SQLException e1) {
e1.printStackTrace();
}} finally {
JDBCUtils.close(null, null, conn); } }
```

数据库事务

- 事务的ACID(acid)属性
  - 1. 原子性（Atomicity）

    原子性是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。
  - 2. 一致性（Consistency）

    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。
  - 3. 隔离性（Isolation）

    事务的隔离性是指一个事务的执行不能被其他事务干扰，即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。
  - 4. 持久性（Durability）

    持久性是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，接下来的其他操作和数据库故障不应该对其有任何影响

数据库事务

- 以第一个 DML 语句的执行作为开始
- 以下面的其中之一作为结束:
  - **COMMIT 或 ROLLBACK 语句**
  - DDL 或 DCL 语句（自动提交）
  - 用户会话正常结束
  - 系统异常终了

DDL: Data Definition Language 数据定义语言(用来定义数据库结构): create table; alter table; drop table; create index; drop index
DCL: Data Control Language 数据控制语言(用来控制数据库的访问)：grant; revoke; commit; rollback; lock;
DML: Data Manipulation Language 数据操纵语言(用来查询与更新记录): insert; update; delete

COMMIT和ROLLBACK语句的优点

使用COMMIT 和 ROLLBACK语句,我们可以:

- 确保数据完整性。
- 数据改变被提交之前预览。
- 将逻辑上相关的操作分组。

数据完整性:
存储在数据库中的所有数据值均处于正确的状态。如果数据库中存储有不正确的数据值，则该数据库称为已丧失数据完整性。
数据库采用多种方法来保证数据完整性，包括外键、束约、规则和触发器。

提交或回滚前的数据状态

- 改变前的数据状态是可以恢复的
- 执行 DML 操作的用户可以通过 SELECT 语句查询提交或回滚之前的修正
- 其他用户不能看到当前用户所做的改变，直到当前用户结束事务。
- DML语句所涉及到的行被锁定， 其他用户不能操作。

提交后的数据状态

- 数据的改变已经被保存到数据库中。
- 改变前的数据已经丢失。
- 所有用户可以看到结果。
- 锁被释放，其他用户可以操作涉及到的数据。

提交数据

●改变数据

```
DELETE FROM employees
WHERE  employee_id = 99999;
1 row deleted.

INSERT INTO departments
VALUES (290, 'Corporate Tax', NULL, 1700);
1 row inserted.
```

●提交改变

```
COMMIT;
Commit complete.
```

数据回滚后的状态

使用 ROLLBACK 语句可使数据变化失效:

- 数据改变被取消。
- 修改前的数据状态可以被恢复。

```
DELETE FROM copy_emp;
22 rows deleted.
ROLLBACK;
Rollback complete.
```

附：数据库的隔离级别

- 对于同时运行的多个事务，当这些事务访问数据库中相同的数据时，如果没有采取必要的隔离机制，就会导致各种并发问题：
  - 脏读：对于两个事务 T1, T2, T1 读取了已经被 T2 更新但还没有被提交的字段。之后，若 T2 回滚，T1读取的内容就是临时且无效的。
  - 不可重复读：对于两个事务T1, T2, T1 读取了一个字段，然后 T2 更新了该字段。之后，T1再次读取同一个字段，值就不同了。
  - 幻读：对于两个事务T1, T2, T1 从一个表中读取了一个字段，然后 T2 在该表中插入了一些新的行。之后，如果 T1 再次读取同一个表，就会多出几行。
- **数据库事务的隔离性：**数据库系统必须具有隔离并发运行各个事务的能力，使它们不会相互影响，避免各种并发问题。
- 一个事务与其他事务隔离的程度称为隔离级别. 数据库规定了多种事务隔离级别，不同隔离级别对应不同的干扰程度，隔离级别越高，数据一致性就越好，但并发性越弱。

附：数据库的隔离级别

- 数据库提供的 4 种事务隔离级别:

隔离级别	描述
READ UNCOMMITTED (读未提交数据)	允许事务读取未被其他事物提交的变更. 脏读, 不可重复读和幻读的问题都会出现
READ COMMITED (读已提交数据)	只允许事务读取已经被其它事务提交的变更. 可以避免脏读, 但不可重复读和幻读问题仍然可能出现
REPEATABLE READ (可重复读)	确保事务可以多次从一个字段中读取相同的值. 在这个事务持续期间, 禁止其他事物对这个字段进行更新. 可以避免脏读和不可重复读, 但幻读的问题仍然存在.
SERIALIZABLE(串行化)	确保事务可以从一个表中读取相同的行. 在这个事务持续期间, 禁止其他事务对该表执行插入, 更新和删除操作. 所有并发问题都可以避免, 但性能十分低下.

- Oracle 支持的 2 种事务隔离级别：**READ COMMITED**, SERIALIZABLE。
  Oracle 默认的事务隔离级别为: READ COMMITED
- Mysql 支持 4 种事务隔离级别. Mysql 默认的事务隔离级别为:
  REPEATABLE READ

附：在 MySql 中设置隔离级别

- 每启动一个 mysql 程序, 就会获得一个单独的数据库连接. 每个数据库连接都有一个全局变量 @@tx_isolation, 表示当前的事务隔离级别.
- 查看当前的隔离级别: SELECT @@tx_isolation;
- 设置当前 mySQL 连接的隔离级别:
  - set transaction isolation level read committed;
- 设置数据库系统的全局的隔离级别:
  - set **global** transaction isolation level read committed;

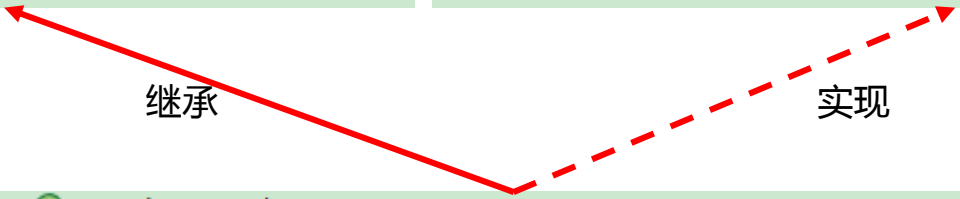
SET autocommit = 0; 禁止操作自动提交

父类

- BaseDao<T>
  - queryRunner : QueryRunner
  - type : Class<T>
  - BaseDao()
  - update(Connection, String, Object...) : int
  - getBean(Connection, String, Object...) : T
  - getBeanList(Connection, String, Object...) : List<T>
  - getValue(Connection, String, Object...) : Object

接口

- BookDao
  - getBooks() : List<Book>
  - saveBook(Book) : void
  - deleteBookById(String) : void
  - getBookById(String) : Book
  - updateBook(Book) : void
  - getPageBooks(Page<Book>) : Page<Book>
  - getPageBooksByPrice(Page<Book>, double, double) : Page<Book>



- BookDaoImpl
  - getBooks() : List<Book>
  - saveBook(Book) : void
  - deleteBookById(String) : void
  - getBookById(String) : Book
  - updateBook(Book) : void
  - getPageBooks(Page<Book>) : Page<Book>
  - getPageBooksByPrice(Page<Book>, double, double) : Page<Book>

具体实现

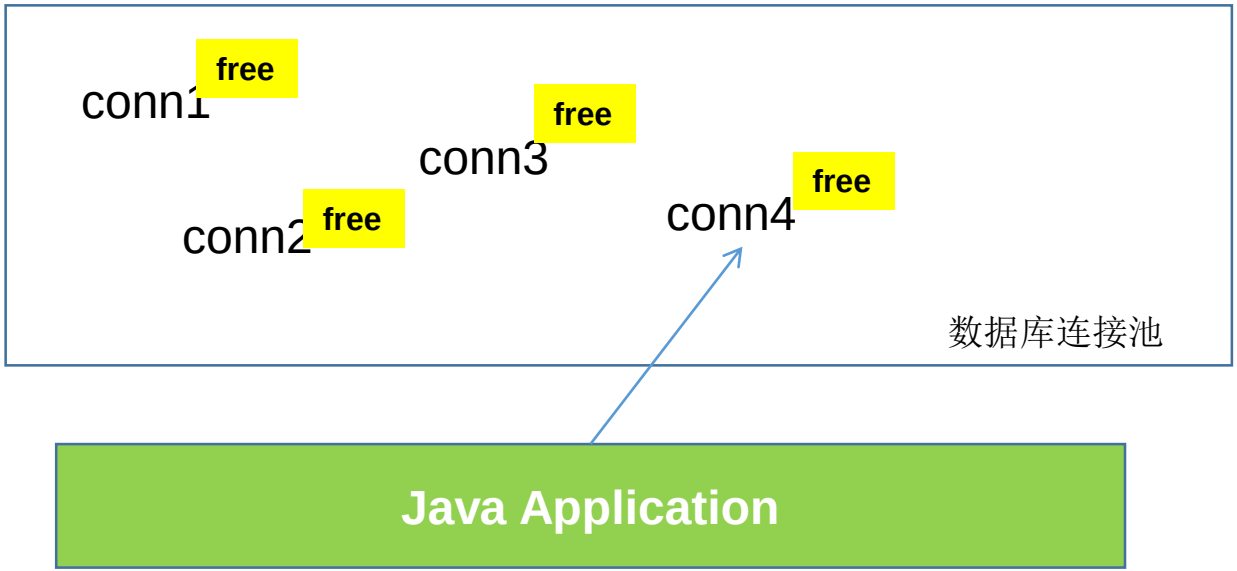
6-数据库连接池

JDBC数据库连接池的必要性

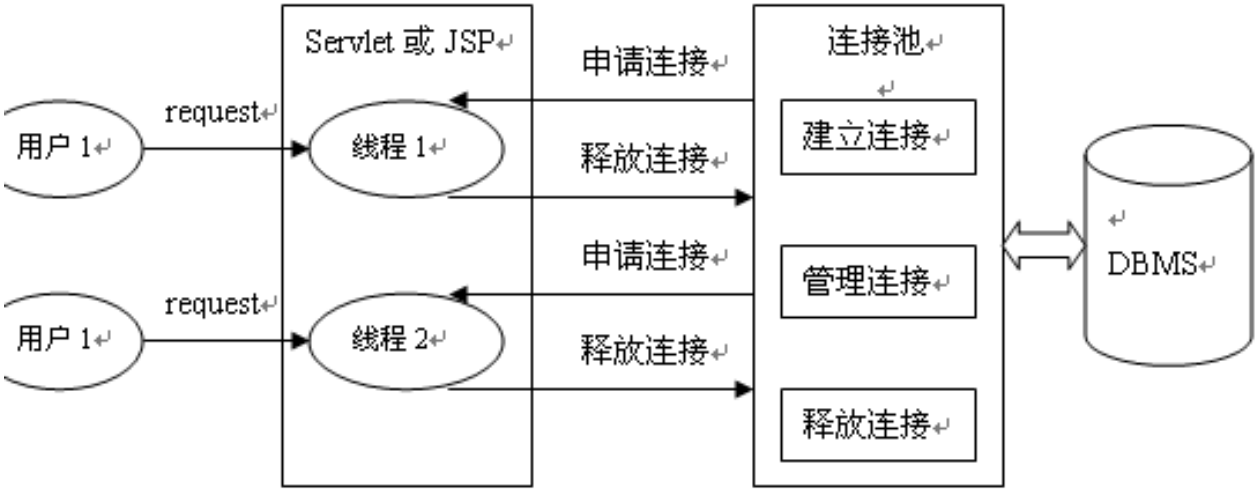
- 在使用开发基于数据库的web程序时，**传统的模式**基本是按以下步骤：
  - 在主程序（如servlet、beans）中建立数据库连接
  - 进行sql操作
  - 断开数据库连接
- 这种模式开发，存在的问题:
  - 普通的JDBC数据库连接使用 DriverManager 来获取，每次向数据库建立连接的时候都要将 Connection 加载到内存中，再验证用户名和密码(得花费0.05s～1s的时间)。需要数据库连接的时候，就向数据库要求一个，执行完成后再断开连接。这样的方式将会消耗大量的资源和时间。**数据库的连接资源并没有得到很好的重复利用.**若同时有几百人甚至几千人在线，频繁的进行数据库连接操作将占用很多的系统资源，严重的甚至会造成服务器的崩溃。
  - 对于每一次数据库连接，使用完后都得断开。否则，如果程序出现异常而未能关闭，将会导致数据库系统中的内存泄漏，最终将导致重启数据库。
  - 这种开发不能控制被创建的连接对象数，系统资源会被毫无顾及的分配出去，如连接过多，也可能导致内存泄漏，服务器崩溃。

数据库连接池（connection pool）

- 为解决传统开发中的数据库连接问题，可以采用数据库连接池技术。
- 数据库连接池的**基本思想**就是为数据库连接建立一个“缓冲池”。预先在缓冲池中放入一定数量的连接，当需要建立数据库连接时，只需从“缓冲池”中取出一个，使用完毕之后再放回去。
- **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，而不是重新建立一个**。
- 数据库连接池在初始化时将创建一定数量的数据库连接放到连接池中，这些数据库连接的数量是由**最小数据库连接数来设定**的。无论这些数据库连接是否被使用，连接池都将一直保证至少拥有这么多的连接数量。连接池的**最大数据库连接数量**限定了这个连接池能占有的最大连接数，当应用程序向连接池请求的连接数超过最大连接数量时，这些请求将被加入到等待队列中。


free
conn1
free
conn3
free
conn2
conn4
数据库连接池
Java Application

数据库连接池的工作原理



数据库连接池技术的优点

- **资源重用**
  - 由于数据库连接得以重用，避免了频繁创建，释放连接引起的大量性能开销。在减少系统消耗的基础上，另一方面也增加了系统运行环境的平稳性。
- **更快的系统反应速度**
  - 数据库连接池在初始化过程中，往往已经创建了若干数据库连接置于连接池中备用。此时连接的初始化工作均已完成。对于业务请求处理而言，直接利用现有可用连接，避免了数据库连接初始化和释放过程的时间开销，从而减少了系统的响应时间
- **新的资源分配手段**
  - 对于多应用共享同一数据库的系统而言，可在应用层通过数据库连接池的配置，实现某一应用最大可用数据库连接数的限制，避免某一应用独占所有的数据库资源
- **统一的连接管理，避免数据库连接泄露**
  - 在较为完善的数据库连接池实现中，可根据预先的占用超时设定，强制回收被占用连接，从而避免了常规数据库连接操作中可能出现的资源泄露

两种开源的数据库连接池

- JDBC 的数据库连接池使用 javax.sql.DataSource 来表示，DataSource 只是一个接口，该接口通常由服务器(Weblogic, WebSphere, Tomcat)提供实现，也有一些开源组织提供实现：
  - **DBCP 数据库连接池**
  - **C3P0 数据库连接池**
- DataSource 通常被称为数据源，它包含连接池和连接池管理两个部分，习惯上也经常把 DataSource 称为连接池
- **DataSource用来取代DriverManager来获取Connection，获取速度快，同时可以大幅度提高数据库访问速度。**

C3P0 数据源

```
ComboPooledDataSource ds1 = null;
ds1 = new ComboPooledDataSource();
try {
    ds1.setDriverClass("com.mysql.jdbc.Driver");
    ds1.setJdbcUrl("jdbc:mysql://localhost:3309/test");
    ds1.setUser("root");
    ds1.setPassword("1230");
    ds1.setMaxPoolSize(40);
    ds1.setMinPoolSize(2);
    ds1.setInitialPoolSize(10);

    Connection conn1 = ds1.getConnection();
    System.out.println(conn1);

} catch (Exception e) {
    e.printStackTrace();
}
```

DBCP 数据源

- DBCP 是 Apache 软件基金组织下的开源连接池实现，该连接池依赖该组织下的另一个开源系统：Common-pool. 如需使用该连接池实现，应在系统中增加如下两个 jar 文件：
  - Commons-dbcp.jar：连接池的实现
  - Commons-pool.jar：连接池实现的依赖库
- Tomcat 的连接池正是采用该连接池来实现的。该数据库连接池既可以与应用服务器整合使用，也可由应用程序独立使用。

DBCP 数据源使用范例

- 数据源和数据库连接不同，数据源无需创建多个，它是产生数据库连接的工厂，因此整个应用只需要一个数据源即可。
- 当数据库访问结束后，程序还是像以前一样关闭数据库连接：conn.close(); 但上面的代码并没有关闭数据库的物理连接，它仅仅把数据库连接释放，归还给了数据库连接池。

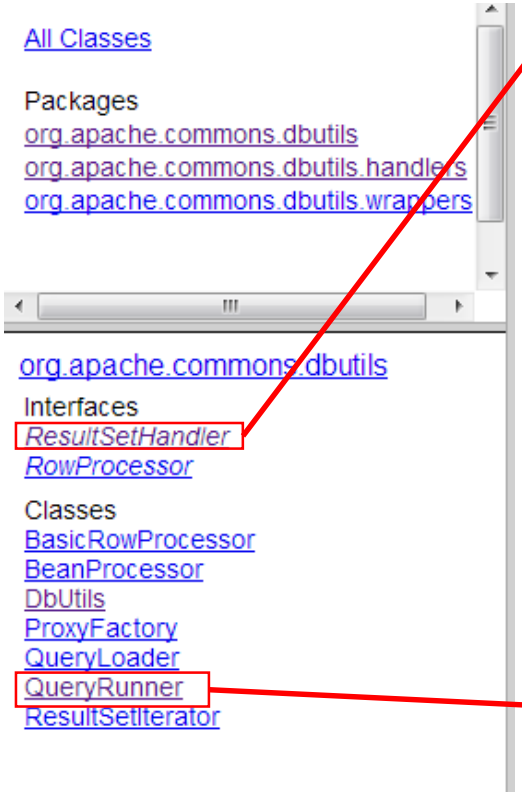
范　例

```
BasicDataSource ds = null;
//创建数据源对象
ds = new BasicDataSource();
//设置连接数据库的 驱动
ds.setDriverClassName("com.mysql.jdbc.Driver");
//设置连接数据库的 url
ds.setUrl("jdbc:mysql://localhost:3309/test");
//设置连接数据库的 用户名
ds.setUsername("root");
//设置连接数据库的 密码
ds.setPassword("1230");
//设置数据库连接池的 初始连接数
ds.setInitialSize(5);
//设置连接池最多可有多少个活动连接数
ds.setMaxActive(20);
//设置连接池中最少有 2 个空闲连接
ds.setMinIdle(2);

Connection conn = null;
try {
    conn = ds.getConnection();
    System.out.print(conn);
} catch (SQLException e) {
    e.printStackTrace();
}
```

7-DBUtils工具类

将常用的操作数据库的JDBC的类和方法集合在一起，就是DBUtils.



ResultSetHandler，此接口用于处理数据库查询操作得到的结果集。不同的结果集的情形，由其不同的子类来实现

QueryRunner，提供数据库操作的一系列重载的update()和query()操作

org.apache.commons.dbutils

Interface ResultSetHandler<T>

All Known Implementing Classes:

AbstractKeyedHandler, AbstractListHandler, ArrayHandler, ArrayListHandler, BeanHandler, BeanListHandler, ColumnListHandler, KeyedHandler, MapHandler, MapListHandler, ScalarHandler

---

BeanHandler:把结果集转为一个 Bean

BeanListHandler:把结果集转为一个 Bean 的集合

MapHandler:把结果集转为一个 Map

MapListHandler:把结果集转为一个 Map 的 List

ScalarHandler:把结果集转为一个类型的数据返回, 该类型通常指 String 或其它 8 种基本数据类型.


尚硅谷