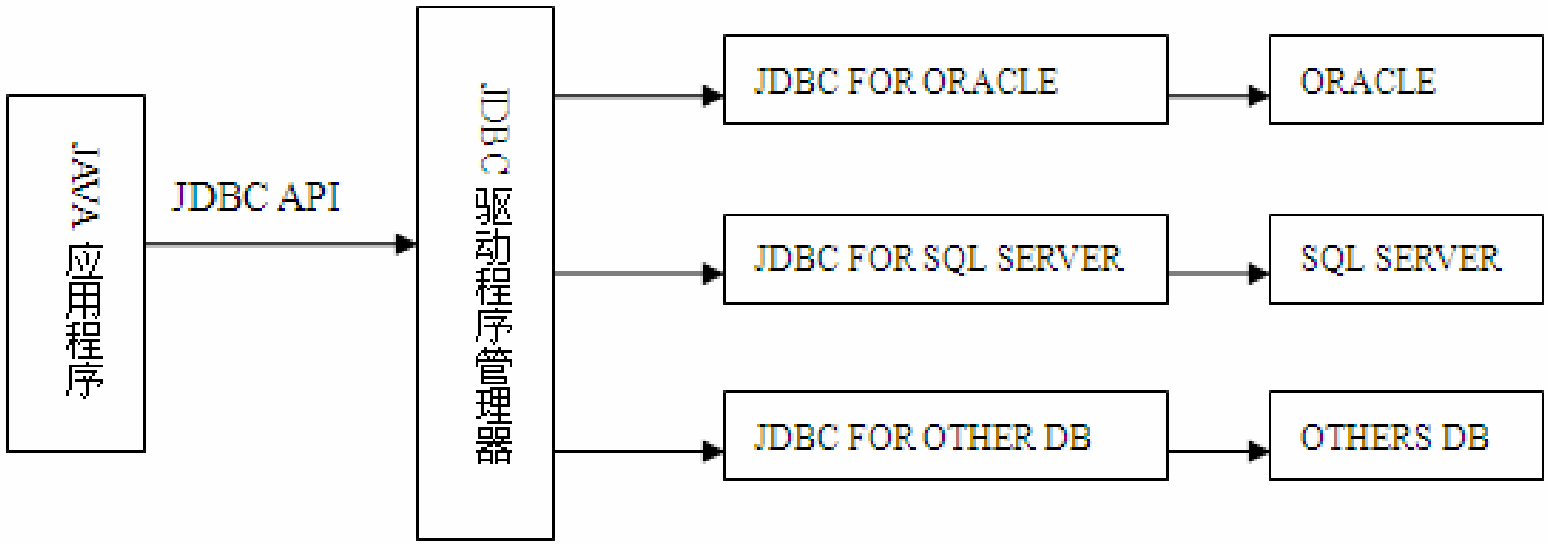
JDBC

Java数据库连接

JDBC示意图


JAVA 应用程序
JDBC API
JDBC 驱动程序管理器
JDBC FOR ORACLE
ORACLE
JDBC FOR SQL SERVER
SQL SERVER
JDBC FOR OTHER DB
OTHERS DB

使用JDBC之前的准备

- java中操作数据库相关的API未实现的，真正实现操作的是数据库厂家提供的类库

  oracle提供的classes12.jar ,ojdbc14.jar

  sql server提供的mssqlserver.jar等

JDBC应用步骤1：注册驱动程序

- 方法1
  - Class.forName(“驱动完整类名")
  - 例如
  - Class.forName("oracle.jdbc.driver.OracleDriver");
- 方法2
  - DriverManager.registerDriver(驱动类对象);
  - 例如
  - DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

JDBC应用步骤2：建立连接

- 使用DriverManager.getConnection方法
- 例如：

```
String url="jdbc:oracle:thin:@127.0.0.1:1521:orcl";
String user = "scott";
String pwd = "tiger";
Connection conn = DriverManager.getConnection(url,user,pwd);
```

JDBC应用步骤3：创建语句对象

- 例如

```
Statement stmt = conn.createStatement();
```

JDBC应用步骤4：执行SQL语句

- 例如

```
ResultSet rs = stmt.executeQuery("select * from employees");
```

JDBC应用步骤5：遍历返回的结果集

- 例如

```
while (rs.next()){
    System.out.println("编号:" + rs.getInt(1) + "\t姓名：" +
    rs.getString(3));
}
```

如果是非查询操作，例如DML操作则无此步骤

JDBC应用步骤6：关闭对象释放资源

- 例如

```
rs.close();
stmt.close();
conn.close();
```

数据类型之间的映射关系

SQL类型	**JAVA类型**
CHAR	String
REAL	Float
DATALINK	java.net.URL
REF	Ref
STRUCT	Struct
DISTINCT	Mappingofunderlyingtype
ARRAY	Array
BLOB	Blob
CLOB	Clob
TIMESTAMP	java.sql.Timestamp
TIME	java.sql.Time
DATE	java.sql.Date
LONGVARBINARY	byte[]
VARBINARY	byte[]
BINARY	byte[]
DOUBLE	double
FLOAT	double
VARCHAR	String
LONGVARCHAR	String

Statement对象执行SQL语句的两种方法

- executeQuery()
  - 执行查询单个结果集的语句，该语句返回单个 ResultSet 对象
- executeUpdate()
  - 执行给定 SQL 语句，该语句可能为 INSERT、UPDATE 或 DELETE 语句，或者不返回任何内容的 SQL 语句（如 SQL DDL 语句），该语句返回影响的行数，如果是DDL语句则返回0

可滚动的结果集游标指针

- Connection对象创建Statement对象有两种形式：
  - ✓一种是用createStatement()无参方法，这种方法创建的Statement对象执行的查询的结果集指针只能向下移动
  - ✓另一种方法是用createStatement(int resultSetType, int resultSetConcurrency)有参方法，这种方法创建的Statement对象执行的查询结果集的指针是可以任意移动的

可滚动的结果集游标指针（续）

后一种方法有两个参数：分别是结果集游标类型和结果集是否可更新

结果集游标类型有三个设置值：

1. ResultSet.TYPE_FORWARD_ONLY 指针只能向下移动（默认）
2. ResultSet.TYPE_SCROLL_INSENSITIVE 指针可任意滚动，但结果集中被更新的行不可视
3. ResultSet.TYPE_SCROLL_SENSITIVE 指针可任意滚动，结果集中被更新的行可视

结果集是否可更新有两个设置值：

1. ResultSet.CONCUR_READ_ONLY 只读（默认）
2. ResultSet.CONCUR_UPDATABLE可更新记录

关于指针移动的方法

默认打开结果集的时候指针位于第一行记录的上面

next() 下一行

previous() 上一行

first() 第一行

last() 最后一行

absolute(n) 定位到第n条

relative(n) 相对于当前指针位置定位到第n条，n为正数下移，n为负数上移

beforeFirst() 第一条的前面

afterLast() 最后一条的后面

getRow() 获得当前行的行号

结果集元数据（Meta-Data）

- ResultSet对象的getMetaData()方法返ResultSetMetaData对象包含了结果集信息

```
stmt = conn.createStatement();
rs = stmt.executeQuery("select * from employees");
//获取元数据
ResultSetMetaData rsMeta = rs.getMetaData();
//打印列的个数
System.out.println("列数：" + rsMeta.getColumnCount());
//第1列列名
System.out.println(rsMeta.getColumnName(1));
//第1列数据类型
System.out.println(rsMeta.getColumnTypeName(1));
```

预编译SQL语句对象 PreparedStatement

- 继承了Statement，特点是预编译好SQL语句并存放在PreparedStatement对象中，适合在SQL语句中具有一个或多个输入参数，用?做占位符，在语句执行之前用setXXX方法给参数赋值

```
pstmt = conn.prepareStatement("insert into student values (?,?)");
pstmt.setInt(1, 101);
pstmt.setString(2, "tom");
```

已存储过程 CallableStatement

- CallableStatement 对象为所有的 DBMS 提供了一种以标准形式调用已储存过程的方法。
- 已储存过程储存在数据库中。对已储存过程的调用是CallableStatement对象所含的内容。这种调用是用一种换码语法来写的，有两种形式：一种形式带结果参，另一种形式不带结果参数。结果参数是一种输出 (OUT) 参数，是已储存过程的返回值。两种形式都可带有数量可变的输入（IN 参数）、输出（OUT 参数）或输入和输出（INOUT 参数）的参数。问号将用作参数的占位符。
- 使用已存储过程的好处是存储过程在数据库中就已经编译了，执行速度相对会快一些。

CallableStatement使用语法

调用无返回值存储过程的语法：

CallableStatement cstmt = c.prepareCall(“{call 过程名[(?, ?, ...)]}”);

调用有返回值存储过程（函数）的语法

CallableStatement cstmt = c.prepareCall(“{? = call 过程名[(?, ?, ...)]}”);

批处理

- 当连续执行多条SQL语句时，正常的执行顺序是将SQL语句每条语句发送一次到数据库，而利用批处理就可以将多条SQL语句增加到批处理中，一次性将多条SQL语句发送到数据库
- **addBatch();//** 加入到批处理
- **executeBatch();//** 执行批处理

数据库事务

- 和数据库事务相关的方法：
  - conn. setAutoCommit() 设置是否自动提交，默认是true
  - conn.rollback() 回退事务
  - conn.commit() 提交事务
  - 需要注意的是**conn.close()**会自动提交未提交的事务

DAO模式

- (Data Access Object)模式是将数据访问，例如增删改查等操作和业务代码分离开，实现一个单独的数据访问层，提供给使用者数据访问的接口，使用者无需知道数据访问的细节，只要调用适当的方法即可，是一种黑箱操作

- 提供一个DAO工厂类
- 提供一个DAO接口
- 提供一个实现了DAO接口的具体类
- 提供数据传输对象DTO

JNDI和数据库连接池

- 连接池的好处
  - 对于一个简单的数据库应用，由于对于数据库的访问不是很频繁。这时可以简单地在需要访问数据库时，就新创建一个连接，用完后就关闭它，这样做也不会带来什么明显的性能上的开销。但是对于一个复杂的数据库应用，情况就完全不同了。频繁的建立、关闭连接，会极大的减低系统的性能
  - 如果创建了数据库连接池，连接池在初始化的时候就会实现创建好N个数据库连接，当有程序连接数据库时会从连接池拿出一个连接，使用完之后再放回去，使得连接可以重复使用。

JNDI和数据库连接池

- 关于**JNDI**
  - **Java Naming and Directory Interface**，一组帮助做多个命名和目录服务接口的API。
  - 是建立在一个部署服务器上，事先做好数据库的连接包括连接数据库的url，用户名，密码等等，然后形成一个服务，java程序可以直接使用这个服务来连接数据库。

JNDI配置context.xml文件

```
<?xml version="1.0" encoding="utf-8"?>
<Context>
    <Resource name="jdbc/test"
                        auth="Container"
                        type="javax.sql.DataSource"
                        maxActive="100"
                        maxIdle="30"
                        maxWait="10000"
                        username="scott"
                        password="tiger"
                        driverClassName="oracle.jdbc.driver.OracleDriver"
                        url="jdbc:oracle:thin:@127.0.0.1:1521:orcl" />
</Context>
```

context.xml解释

在xml文件中Context标签中只有一个Resource子标签，里面的属性设置如下：

- **name** 自定义的命名，可以自己随便起名字，名字可以用多个斜杠分隔，名字要和web.xml文件中 <res-ref-name>XXX</res-ref-name> 一样
- **auth** 管理者固定是Container－容器管理
- **type** 类型固定是"javax.sql.DataSource"
- **maxActive** 在同一时刻最大的活动连接数，就是最大连接数，设置为0表示无限制
- **maxIdle** 在同一时刻最大的非活动连接数，就是可以保留多少空闲连接，超过这个数目，数据库连接将被标记为不可用，资源释放。比如说值为20，表示即使没有数据库连接时依然可以保持20空闲的连接，而不被清除，随时处于待命状态。
- **maxWait** 如果没有闲置的连接，并且连接数量已经到达了maxActive的极限，那么等待多久（毫秒）抛出异常
- **username** 数据库用户名
- **password** 数据库密码
- **url** 数据库连接url
- **driverClassName** 数据库连接驱动类的完整类名

web.xml 配置

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

  <resource-ref>
      <description>jndi dbcp test</description>
      <res-ref-name>jdbc/test </res-ref-name>
      <res-type>javax.sql.DataSource</res-type>
      <res-auth>Container</res-auth>
  </resource-ref>
</web-app>
```

web.xml 配置解释

- <resource-ref>
- <descrtiption>引用资源说明，随便写点描述</descrtiption>
- <res-ref-name>引用资源的JNDI名</res-ref-name>
- <res-type>引用资源的类名</res-type>
- <res-auth>管理者（Container）</res-auth><!--Container－容器管理 Application－Web应用管理-->
- </resource-ref>

应用JNDI数据库连接池

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ page import="java.sql.*"%>
<%@ page import="javax.sql.DataSource" %>
<%@ page import="javax.naming.InitialContext" %>
<%
	//初始化查找命名空间
	InitialContext context = new InitialContext();
	//找到DataSource,对名称进行定位java:comp/env是必须加的,后面跟你的DataSource名
	DataSource ds = (DataSource) context
				.lookup("java:comp/env/jdbc/test");
	//获得连接对象
	Connection conn = ds.getConnection();
	//查询数据
	Statement stmt = conn.createStatement();
	String sql = "select * from employees";
	ResultSet rs = stmt.executeQuery(sql);
	while (rs.next()) {
		System.out.println(rs.getString(1) + "," + rs.getString(2));
	}
	rs.close();
	stmt.close();

	//这不是关闭，这是把连接对象还给连接池
	conn.close();
%>
```

JDBC结束，谢谢