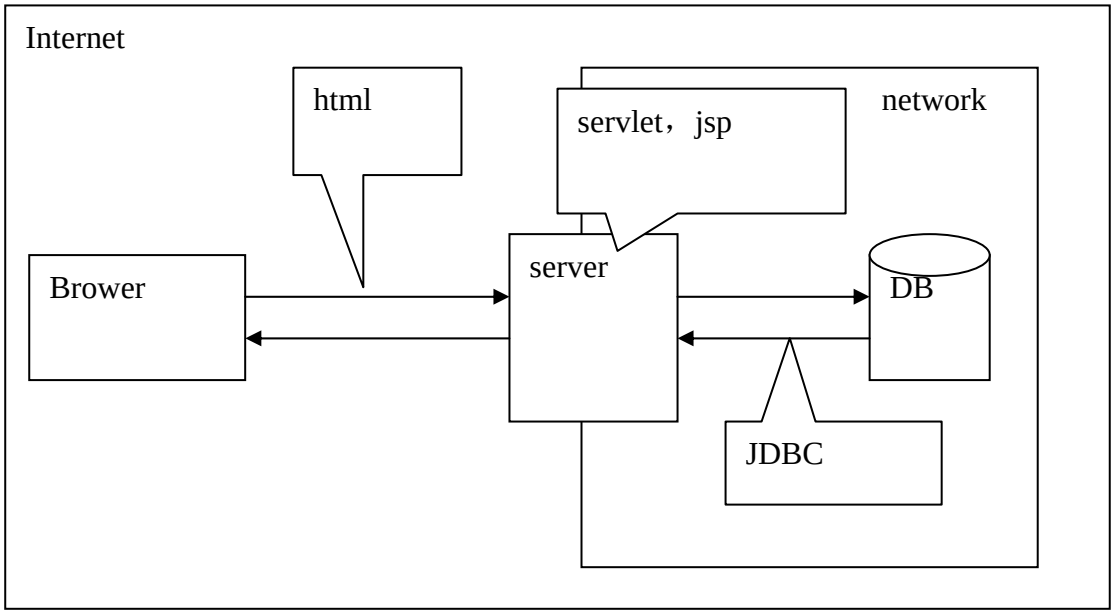
1. JDBC 笔记（06－10－12 更新）未完继续更新中

1.1. JDBC 在 web 中的结构图：



为了提高扩张性，可维护性，所以使用架构。
struts （MVC 架构） 开源的
hibernate （XML）
spring (XML)
比较 collection 使用了大量的接口，实现类只用在 new 上一点，其他都用的是接口！

1.2. JDBC 介绍

1.1.1. 连接到数据库的方法

1) ODBC(Open Database Connectivity)
一个以 C 语言为基础访问 SQL 为基础数据库引擎的接口，它提供了一致的接口用于和数据库沟通以及访问数据。
2) JDBC
Java 版本的 ODBC

1.1.2. JDBC 应用编程接口

答：JDBC 应用编程接口是：

1) 标准的数据访问接口，可以连到不同的数据库的规范；
2) JAVA 编程语言的一组类和接口。

JDBC 应用编程接口能够：

1) 连接到数据库;
2) 发 SQL 查询字符串到数据库;
3) 处理结果。

JDBC 应用编程接口有二个主要的部分：

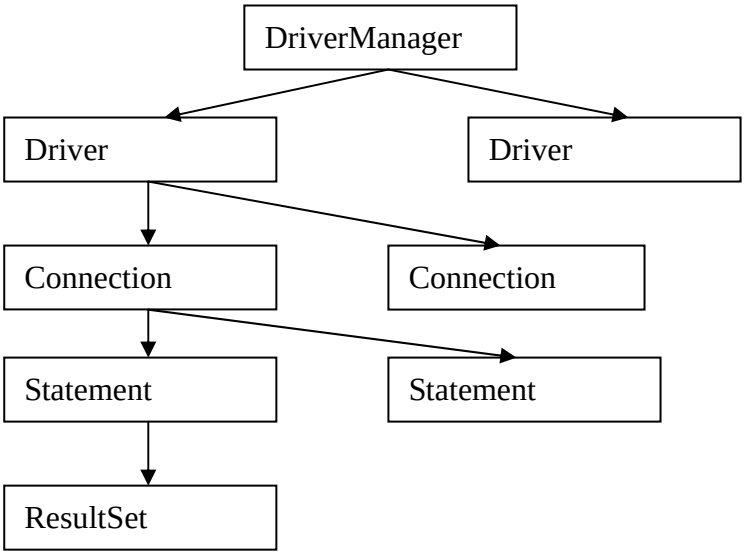
1) JAVA 应用程序开发接口面向 JAVA 应用程序开发者;
2) JDBC 驱动程序开发接口，面系数据库驱动开发者；

按包又分为：

java.sql 为 Java SE 标准开发环境－－－－JDBC 2.0 规范
javax.sql 为 Java EE 扩展开发环境。

java.sql 中的类和接口：

Driver
DriverManager
Connection
Statement
PreparedStatement
CallableStatement
ResultSet
DatabaseMetadata
ResultMetadata
Types



1.1.3. JDBC 开发者接口

答：1) java.sql--java 2 平台下 JDBC 的主要功能，标准版(J2SE)
2) javax.sql--java 2 平台下 JDBC 增强功能，企业版(J2EE)

1.1.4. JDBC Driver

答：1) 一大堆实现了 JDBC 类和接口的类;
2) 提供了一个实现 java.sql.Driver 接口的类。

1.1.5. JDBC Driver 的四种类型

答：1) JDBC-ODBC 桥
由 ODBC 驱动提供 JDBC 访问
2) 本地 API
部分 Java driver 把 JDBC 调用转化成本地的客户端 API
3) JDBC-net
纯的 Java driver，将 JDBC 调用转入 DBMS，与网络协议无关。然后通过服务器将调用转为 DBMS 协议。
4) 本地协议
纯的 java driver，将 JDBC 调用直接转为 DBMS 使用的网络协议

8. 创建一个基本的 JDBC 应用

答：1) 步骤一：注册一个 driver;
2) 步骤二：建立一个到数据库的连接;
3) 步骤三：创建一个 statement;
4) 步骤四：执行 SQL 语句;
5) 步骤五：处理结果;
6) 步骤六：关闭 JDBC 对象

9. 注册一个 Driver(步骤一)

答：1) driver 被用于连接到数据库;
2) JDBC 应用编程接口使用第一个能成功连接到给定 URL 的 driver;
3) 在同一时间可以装载多个 driver

注册一个 driver 的方法： 使用类 loader(装载;实例化;注册入 DriverManager)
a. Class.forName("Com.ibm.db2.jdbc.app.DB2Driver");
b. Class.forName("Com.ibm.db2.jdbc.net.DB2Driver");
c.Class.forName("Com.microsoft.jdbc.sqlServer.SQLServerDriver);
d. Class.forName("oracl.jdbc.driver.OracleDriver");
e. Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

Class.forName(“.....driver”);

DriverManager.getConnection(“”);
为什么能加载驱动？
答：JDBC 规范中的要求

```
public class OracleDriver implements Driver
{
    //利用静态代码块注册驱动
    static
    {
        DriverManager.riegisterDriver(new OracleDriver());
    }
}
```

1.3. 使用 JDBC

1.1.6. JDBC 连接数据库操作

1) 步骤一：注册一个 driver;
2) 步骤二：建立一个到数据库的连接;
3) 步骤三：创建一个 statement;
4) 步骤四：执行 SQL 语句;
5) 步骤五：处理结果;
6) 步骤六：关闭 JDBC 对象

Oracle , Sybase , SqlServer ,DB2 等厂商提供驱动（Driver）

1.1.7. 建立一个数据库的连接

1. Driver driver=new OracleDriver();
   Connection con=driver.connection(url:String,info:Perperties)
2. Class.forName(“oracle.jdbc.driver.OracleDriver”);

DriverManager.getConnection(url:String,username:String,passwd:String);

为什么能通过 Class.forName()加载驱动：
JDBC 规范中要求 Driver 类是这样的，

```
package oracle.jdbc.driver;
public class OracleDriver implements Driver
{
    //利用静态代码块注册驱动
    static
    {
        DriverManager.registerDriver(new OracleDriver());
    }
    Connection connection(url,username,password)
```

```
        {
        return 实现类;
        }
}
```

1.1.8. Connection 接口的性能优化

1、设置适当的参数

DriverManager.getConnection(String url,Properties props);

例如:

```
Properties props=new Properties();
props.put("user","wuwei");
props.put("password","wuwei");
props.put("defaultRowPrefectch","30");
props.put("dufaultBatchValue","5");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@hostsString",props);
```

对象可以通过设置setDefaultRowPrefetch(int) 和 setDefaultBatchValue(int)两个参数类优化连接

2、使用连接池

可以自己写一个连接池，这样程序的灵活性强，便于移植。

apache 项目开发了一套非常通用而表现非常稳定的对象池 http://jakarta.apache.org/commons/pool.htm

注意几点：对象池里有没有回收机制，对象池里有机有容量限制，对象池里有多少个闲置对象(可以释放)

1.1.9. 创建一个 statement

```
Statement st=con.createStatement();
Statement st=con.createStatement(int resultSetType,int
                                    resultSetConcurrency)
PreparedStatement ps=con.perpareStatement();
CallableStatement cs=con.prepareCall();
```

1.1.10. 执行 SQL 语句

```
Statement st=.....
ResutlSet rs=st.executeQuery(“sql”);
st.executeUpdate(“sql”);
```

1.1.11. 处理结果

```
ResulteSet rs.....
rs.next();
rs.close();
```

1.1.12. 关闭 JDBC 对象

```
rs.close();
ps.close();
con.close();
```

1.1.13. 一个完整的 JDBC 程序

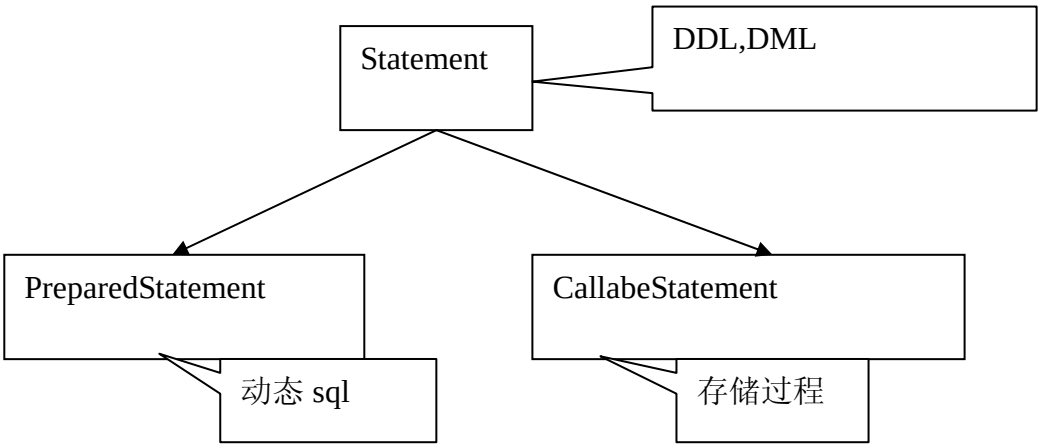
```
package mudi.jdbc;
import java.sql.*;
class ResultSetTest
{
    public static void main(String[] args)throws Exception
    {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection
        conn=DriverManager.getConnection("jdbc:oracle:thin:@10.10.3.237:1521:ta
        rena","nanjing","nanjing");
        Statement
        st=conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultS
        et.CONCUR_UPDATABLE);
        ResultSet rs=st.executeQuery("select * from users");
        System.out.println("test");
        while(rs.next()){
            System.out.print(“id:”+rs.getInt(1));
            System.out.print(“name:”+rs.getString(2));
            System.out.print(“age:”+rs.getInt(3));
        }
    }
}
```

1.4. JDBC 高级功能

1.1.14. 项目中关于代码复用的知识

在 JDBC 编程中会出现大量重复代码
采取复用
继承复用
聚合复用
工具方法复用－－－static 方法

1.1.15. Statement 的结构和用途



注意： 程序的单线程和多线程在设计时的区别。PreparedStatement 需要数据库支持预编译功能

1.1.16. Exceptions

JDBC 的各种错误是由 java.sql.SQLException,SQLWarning 表现的
SQLException 异常发生在
JDBC 连接失败
SQL 语法错误
使用了数据库不支持的功能
访问了一个不存在的列

SQLException 可以接受后重新定义抛出

SQLWarning 发生在
产生一个非致命 SQL 状态

1.1.17. 元数据

答：关于数据的信息，例如类型或者容量。通过 JDBC API 可以访问：

1) 数据库元数据;
   a. 使用 connection.getMetadata 方法返回 DataMetaData 引用
   b. 能够使用 isReadOnly 等类方法获取数据库信息
2) 结果集元数据;
   a. 使用 ResultSet.getMetadata 方法返回 ResultSetMetaData 引用
   b. 能够使用 getColumnCount 等类方法获取结果集信息

数据库元数据的代码：

```
class    DataBaseUtil
{
     public static void main(String[] args)
     {
          Class.forName("oracle.jdbc.driver.OracleDriver");
          Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@192.168.0.39:1521:tarenadb"
               ,"nanjing","nanjing");
          DataBaseMetaData dbmd=con.getMetaData();
          System.out.println(getDefaultTransactionIsolation());
          //得到数据库隔离级别
     }
}
```

结果集元数据的代码：

```
import java.sql.*;

public class ResultSetUtil
{
     public static void printRS(ResultSet rs) throws SQLException
     {
          ResultSetMetaData rsmd = rs.getMetaData();

          while(rs.next())
          {
               for(int i = 1; i <= rsmd.getColumnCount(); i++)
               {
                    String colName = rsmd.getColumnName(i);
                    String colValue = rs.getString(i);
                    if( i > 1)
                    {
                         System.out.print(",");
```

```
                }
                System.out.print(colName + "=" + colValue);
            }
            System.out.println();
        }
    }
}
```

1.1.18. 事务处理

答：1) 一系列的动作作为一个不可分的操作;
2) JDBC API 中使用事务处理步骤：
a. 用 false 作为参数调用 setAutoCommit 方法;
b. 执行一或多个关于数据库的操作;
c. 调用 commit 方法完成改变;
d. 恢复上次提交后的改变，调用 rollback 方法.

```
try
{
    con.setAutoCommit(false);
    Statement stm = con.createStatement();
    stm.executeUpdate("insert into student(name, age, gpa) values('gzhu', 30,
4.8)");
    stm.executeUpdate("insert into student(name, age, gpa) values('liucy', 29,
4.8)");

    stm.commit();
}
catch(SQLException e)
{
    try
    {
        con.rollback();
    }
    catch(Exception e)
    {
    }
}
```

1.1.19. 适当的选择事务的隔离级别

TRANSACTION_READ_UNCOMMITED 性能最高 隔离最低

TRANSACTION_READ_COMMITED 快 。。。。
TRANSACTION_REFEATABLE_READ 中等 。。。。
RANSACTION_SERIALIZABLE 慢 隔离最高

Concurrency Control 并发控制
con.getTransactionIsolation();
con.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITED);

1.5. JDBC2.0 的优点

1.1.20. 结果集的特点（使用 ResultSet 增，删，改）

Scrollabilty 游标控制
向前和向后滚动
绝对和相对游标指定

```
ResultSet rs=stat.createStatement();
rs.absolute(int a);
rs.afterLast();
rs.beforeFirst();
rs.first();
rs.last();
rs.next();
rs.previous();
rs.relative(int);
rs.isAfterLast();
rs.isBeforeFirst();
rs.isFirst();
rs.isLast();
```

更新 ResultSet 一行的代码：

```
/*
步骤：
1．将游标移动到要修改的位置
2．调用 updateXXX()进行修改
3．执行
*/
class  DataBaseUtil
{
    public static void main(String[] args) throws Exception
    {
        Class.forName("org.gjt.mm.mysql.Driver");
```

```
            Connection con = DriverManager.getConnection("
        jdbc:mysql://127.0.0.1:3306/test","nanjing","nanjing");
            DatabaseMetaData dbmd=con.getMetaData();
            System.out.println("TransactionIsolation"+
dbmd.getDefaultTransactionIsolation());
            Statement stat=con.createStatement(
            ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_UPDATAB
        LE);
            ResultSet rs=stat.executeQuery("select * from user");
            rs.absolute(3);
            rs.updateString(2,"Test");
            rs.updateRow();
            rs.close();
            stat.close();
            con.close();
        }
}
```

插入一行代码：

```
/*
```

步骤：

1．移动游标到插入位置

2．将字段依次添加完整

3．执行插入

```
*/
            rs.moveToInsertRow();
            rs.updateInt(1,100);
            rs.updateString(2,"Test");
            rs.updateInt(3,40);
            rs.updateInt(4,0);
            rs.updateString(5,"ShangHai");
            rs.insertRow();
```

1.1.21. Statement 接口的批量处理功能

前提是数据库驱动要支持批量处理

PreparedStatement 会话批量处理的代码：

```
/*
```

步骤：

1．向 Statement 对象中添加 SQL 语句

Statement 里面应该有一个用于存储 SQL 语句的集合。一块执行

2．执行批处理方法

```
*/
import java.sql.*;
class    ResultSetBatchTest
```

```
{
    public static void main(String[] args) throws Exception
    {
        Class.forName("org.gjt.mm.mysql.Driver");
        Connection con =
DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/test"
            ,"nanjing","nanjing");
        con.setAutoCommit(false);
        PreparedStatement stat=con.prepareStatement("insert into User
values(?,?,?,?,?)");
        stat.setInt(1,180);
        stat.setString(2,"JDBC");
        stat.setInt(3,22);
        stat.setInt(4,1);
        stat.setString(5,"beijing");
        stat.addBatch();
        int[] rss=stat.executeBatch();//返回执行成功的语句所影响的行数
        for(int i=0;i<rss.length;i++)
        {
            System.out.println(rss[i]);
        }
        con.commit();
        stat.close();
        con.close();
    }
}
```

Statement 同理

```
con.setAutoCommit(false);
stat.addBatch("insert into User values(id,name,age,sex,address);
stat.executeBatch();
con.commit();
```

1.1.22. Statement 性能的优化

jdbc3 个接口用来处理 sql 的执行,是 Statement PreparedStatement CallableStatement 提供适当的 Statement 接口批量执行 sql 从数据库批量获取数据。

PreparedStatement 比 Statement 性能要好 主要体现在一个 sql 语句多次重复执行的情况。

PreparedStatemnt 只编译解析一次而 Statement 每次编译一次。

批量修改数据库

Statement 提供方法 addBatch(String)和 executeBatch()

调用方法为

```
stmt.addBatch("isnert.....");
stmt.addBatch("update.....");
```

```
stmt.executeBatch();
```

也可以用 PreparedStatement 从而更好的提高性能：

```
pstmt=conn.preparedStatement("insert into test_table(.....) values(....?)");
pstmt.setString(1,"aaa");
pstmt.addBatch();
pstmt.setString(1,"bbb");
pstmt.addBatch();
.....
pstmt.executeBatch();
```

批量地从数据库中取数据：

通过 setFetchSize()和 getFectchSize()方法来设定和查看这个参数，这个参数对体统的性能影响比较大。这个参数太小会严重地降低程序的性能。

Connection Statement ResultSet 都有这个参数,他们对性能地影响顺序是：ResultSet---Statement---Connection

1.1.23. 高级数据类型 SQL3 数据类型

Blob-SQL Blob binary
Clob-SQL Clob charactor
Array-SQL
Structure type
Ref-SQL Ref

Blob-SQL Blob 的使用

SQL 建表语句：

```
CREATE TABLE ImageLibrary
(
    id              number(9) not null,
    name            varchar2(30) not null,
    image               blob,
    constraint PK_Customer primary key(id)
)
```

以读图片为例：

```
/*
同其他数据类型不同的是需要和 Blob 对象建立一个流，将数据输入或者输出。
*/
public class  ImageReaderUtil
{
   public static void main(String[] args) throws Exception
   {
      Class.forName("org.gjt.mm.mysql.Driver");
      Connection                                con                                =
```

```
DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/test"
                ,"nanjing","nanjing");
            Statement stat=con.createStatement();
            ResultSet rs = stat.executeQuery("select * from imagelibrary");
            while(rs.next())
            {
                Blob image=rs.getBlob(3);
                BufferedOutputStream bof=new BufferedOutputStream(
                                new FileOutputStream(rs.getString(2)+".jpg"));
                BufferedInputStream bif=new BufferedInputStream(
                                image.getBinaryStream());
                int c;
                while((c=bif.read())!=-1)
                {
                    bof.write(c);
                }
                bof.close();
                bif.close();
            }
            rs.close();
            stat.close();
            con.close();
        }
}
```

Array-SQL Array 的使用

```
/*
步骤：
1．先建立数据类型
2．再建立表结构
*/
CREATE TYPE HOBBIES as VARRAY(5) of VARCHAR2(20)
CREATE TABLE User_Array
(
    id              number(9),
    username        varchar2(15),
    password        varchar2(15),
    hobbies             HOBBIES,
    constraint PK_User_Array primary key(id)
)
```

在 Statement 中所使用的 sql 语句：

```
String sql = "insert into User_Array(id, username, password, hobbies) values ";
                sql += "(1, 'alan', '123', HOBBIES('swim', 'walking', 'reading', 'skating',
'shooting'))";
```

在 PreparedStatement 中使用的 sql 语句：

```
String sql = "insert into User_Array(id, username, password, hobbies) values ";
            sql += "(1, 'alan', '123', HOBBIES('?', '?', '?', '?', '?'))";
```

Structure 类型的建表语句

```
CREATE TYPE CourseStruct AS object
(
    no          varchar2(10),
    name        varchar2(60),
    fee         number(5)
)

CREATE TABLE Student_Struct
(
    id                  number(9),
    name                varchar2(20),
    attendcourse    CourseStruct,
    constraint PK_Student primary key(id)
)
```

java.sql.Ref ：它是到数据库中的 SQL 结构类型值的引用

如果表结构中有Structure 类型字段，那么在得到的ResultSet中就有对应的Ref对象，因为 SQL 中REF字段的值是一个指向 SQL 结构体类型的逻辑指针，所以在默认情况下 Ref 对象也是一个逻辑指针。因此，在程序中以 Ref 对象的并不能存在实现结构类型的属性。需要从Ref对象得到Struct对象，再依次取得属性值

代码如下：

```
Ref courseRef = (Ref) rs.getObject(1);
Struct course = (Struct) courseRef.getObject();
Object[] courseAttributes = course.getAttributes();
```