



Programming
Your Future

JDBC与JAVA数据库编程



Programming Your Future

课程结构

内容	课时 (H)
第一章: JDBC 的概念	0.5
第二章: JDBC 基础应用	1.5
第三章: JDBC 高级应用	1

第一章： JDBC 的概念

目标：

本章旨在向学员介绍JDBC 的概念，通过本课的学习，学员应该掌握如下知识：

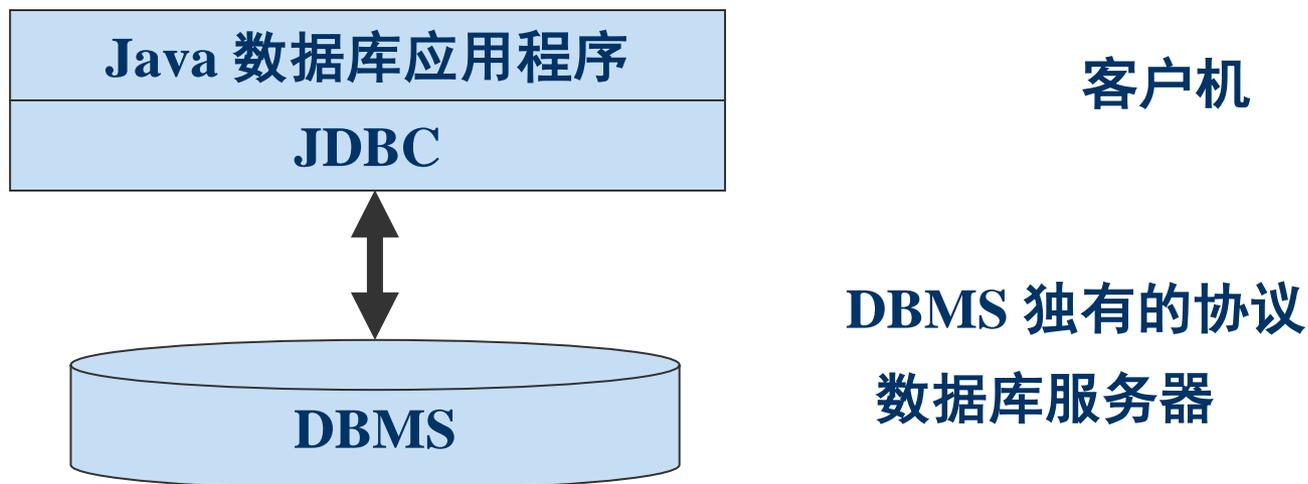
- 1) 了解JDBC的体系结构
- 2) 掌握 java. sql包中常用的基本的 JDBC API

学时：0.5学时

教学方法：讲授ppt

1.1 JDBC 是什么

- JDBC (Java DataBase Connectivity) Java 数据库连接, 主要提供编写 Java 数据库应用程序的 API 支持。



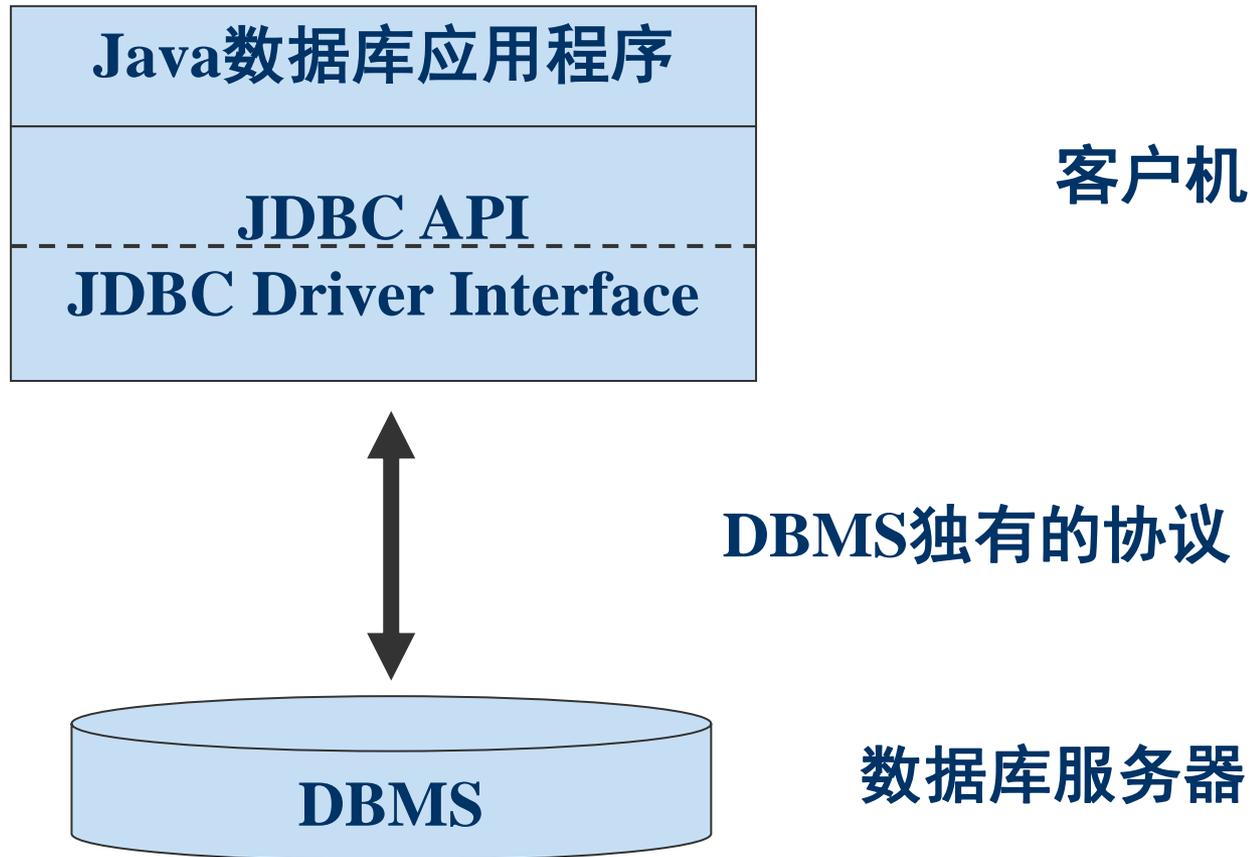
1.2 JDBC 实现的功能

- 创建和管理与数据源的连接
- 发送 SQL 命令至数据源
- 提取并处理由数据源返回至应用程序的结果集

1.3 JDBC 的体系结构

- JDBC的结构可划分为两层：
 - JDBC 驱动程序管理器接口
 - JDBC API

1.4 JDBC体系结构图



1.5 JDBC优缺点

优点：

- (1) JDBC使得编程人员从复杂的驱动器调用命令和函数中解脱出来，可以致力于应用程序中的关键地方。
- (2) JDBC支持不同的关系数据库，这使得程序的可移植性大大加强。
- (3) JDBC API是面向对象的，可以让用户把常用的方法封装为一个类，以备后用。

缺点：

- (1) 使用JDBC，访问数据记录的速度会受到一定程度的影响。
- (2) JDBC结构中包含不同厂家的产品，这就给更改数据源带来了很大的麻烦。

1.6 JDBC版本

JDBC 1.x

- 最初的java连接规范
- 作为一个内插式附件来发布的，不久就被继承到了标准的JDK中
- 提供了基本的数据存储架构、，由一些核心接口组成，包括DriverManager,Connection,Statement,ResultSet.

1.6 JDBC版本

JDBC2.0

- 新特性：可滚动结果集、可更新结果集、批量更新、性能调整。
- Core API:用java.sql包来实现
- Optional Package API:用javax.sql包来实现。

1.6 JDBC版本

JDBC3.0

- 新增了一个保存点的概念，保存点可以用来标记一个事物的某些部分，以便该事物能够回退到一个给定点。
- 新增了控制连接池的更多配置参数。
- 提供了一个迁移路径以便数据库开发商能把他们的JDBC产品朝着java connection体系结构迁移。

1.7 java.sql 包

- java.sql包中定义的常用的基本的 JDBC API:
 - 类 DriverManager—管理一组 JDBC 驱动程序的基本服务
 - 接口 Connection—获得与特定数据库的连接
 - 接口 Statement—用于执行静态 SQL 语句并返回它所生成结果的对象
 - 接口 ResultSet—表示数据库结果集的数据表，通常通过执行查询数据库的语句生成
 - 类 SQLException—有关数据库操作的异常

1.8 小结

- JDBC体系结构
- JDBC API:
 - DriverManager
 - Connection
 - Statement
 - ResultSet
 - SQLException

第二章： JDBC 基础应用

目标：

本章旨在向学员介绍JDBC 的基础应用，通过本课的学习，学员应该掌握如下知识：

1) 掌握JDBC 应用程序的基本步骤的编码

学时：1.5学时

教学方法：讲授ppt
+演示

2.1 创建 JDBC 应用程序的步骤

- 编写 JDBC 应用程序的基本步骤：
 - 导入 JDBC 类或包括 JDBC 类的包
 - 加载 JDBC 驱动程序
 - 建立与数据库的连接
 - 执行 SQL 语句，与数据库交互
 - 关闭连接

2.2 数据库驱动程序

目前的JDBC驱动程序有可以分为以下四大类：

- (1) 采用JDBC-ODBC桥的形式，将JDBC首先翻译为ODBC，然后使用ODBC驱动程序和数据库通信。
 - (2) 由部分JAVA程序和部分本地代码组成，利用开发商提供的本地库函数来直接与数据库通讯。
 - (3) 纯java程序，它使用一种与具体数据库无关的协议将数据库请求发送给一个中间服务器。
 - (4) 纯java的驱动程序，直接与特定的数据库系统通信。直接将jdbc命令转换为数据库系统的本地协议。
- 通常开发中多采用第四种方式，他的使用更加的直接和简便。

2.2 数据库驱动程序

- 各数据库厂商均提供对 JDBC 的支持，即提供数据库连接使用的驱动程序文件
- 需要为数据库应用程序正确加载驱动程序文件以获得数据库连接，实施操作
- Oracle 数据库的 JDBC 驱动程序文件 “classes14.jar”
(Oracle 官方网站下载)

示例

- 在 Eclipse 下创建工程 JdbcOracleTest 及同名主类，为该工程配置新的类库文件指向 classes14.jar。

2.3 加载 JDBC 驱动程序

- Class 类中提供加载驱动程序的方法：

```
public static Class.forName(String className)  
throws ClassNotFoundException
```

className—表示类的描述符的字符串

- Oracle 驱动的类型描述符为：

```
oracle.jdbc.driver.OracleDriver
```

示例:

- 在工程主类 JdbcOracleTest 的 main 方法中增加加载 Oracle 驱动的代码:
 - 声明表示 Oracle 驱动类描述符的字符串变量 driver
 - 调用 Class 类的静态方法 forName 加载该驱动（注意异常处理）

2.4 建立与数据库的连接

- DriverManager 类提供 getConnection 方法可获得指定数据库的连接对象：

```
public static Connection getConnection  
    (String url, String userName, String password)  
    throws SQLException
```

- Oracle 数据库的 url 格式为：

```
jdbc:oracle:thin:@<主机名或IP>:1521:<数据库名>
```

示例

- 修改类 JdbcOracleTest 的 main 方法：
 - 声明表示指定数据库url的字符串变量 url
 - 分别声明表示用户名和口令的字符串变量 userName 和 password，分别初始化为 "SCOTT" 和 "TIGER"
 - 声明Connection接口对象con，赋值为 DriverManager类的getConnection方法的返回值
 - 输出打印“数据库连接成功”的提示信息

2.5 获得 Statement 对象

- Connection 类中提供可获得 Statement 对象的方法:

Statement createStatement() throws SQLException

- 可调用重载的 createStatement 方法，可指定参数，设置数据库操作结果的相关属性。

2.6 执行 SQL 语句

- Statement 类提供可执行 SQL 命令的方法:

```
boolean execute(String sql) throws SQLException
```

```
ResultSet executeQuery(String sql) throws SQLException
```

```
int executeUpdate(String sql) throws SQLException
```

示例（查询）

- 在工程主类 JdbcOracleTest 的 main 方法中增加操作数据库的代码：
 - 获得可发送SQL命令的Statement对象st
 - 调用对象st的executeQuery方法发送SQL查询命令，查询SCOTT下的表DEPT，获得所有记录数据，返回结果集对象rs

示例（增删改查）

- 在工程主类 JdbcOracleTest 的 main 方法中增加操作数据库的代码：
 - 获得可发送SQL命令的Statement对象st
 - 实现对数据库增删改查操作

2.7 操作结果集对象

- ResultSet 类提供可对结果集进行操作的方法：
 - 移动结果集操作指针：

boolean next() throws SQLException

- 指定数据类型根据传入列的名字获取指定列的值：

Xxx getXxx(String columnName) throws SQLException

- 指定数据类型根据传入列的编号获取指定列的值：

Xxx getXxx(1) throws SQLException

2.8 SQL 类型对应 Java 数据类型

SQL Type	Java Type
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int

SQL 类型对应 Java 数据类型 (续)

SQL Type	Java Type
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

2.9 getXxx 方法

Method	Java Technology Type Returned
getASCIIStream	java.io.InputStream
getBigDecimal	java.math.BigDecimal
getBinaryStream	java.io.InputStream
getBoolean	boolean
getByte	byte
getBytes	byte[]
getDate	java.sql.Date
getDouble	double
getFloat	float

getXxx 方法 (续)

Method	Java Technology Type Returned
getInt	int
getLong	long
getObject	Object
getShort	short
getString	java.lang.String
getTime	java.sql.Time
getTimestamp	java.sql.Timestamp
getUnicodeStream	java.io.InputStream of Unicode characters

示例

- 在工程主类 JdbcOracleTest 的 main 方法中增加处理结果集的代码：
 - 以rs对象的next()方法作为while循环的条件，调用对象rs的getXxx方法，指定列名和类型，获取结果集对象中DEPT表的所有数据，并打印输出。

2.10 关闭操作对象及连接

- 可调用类 `ResultSet`、`Statement`、`Connection` 中的关闭方法，立即释放数据库和 JDBC 相关资源：

`void close() throws SQLException`

示例

- 在工程主类 `JdbcOracleTest` 的 `main` 方法中增加关闭数据库操作对象的代码：
 - 关闭结果集对象 `rs`
 - 关闭 `Statement` 对象 `st`
 - 关闭 `Connection` 对象 `con`

2.11 小结

- 编写 JDBC 应用程序的基本步骤：
 - 导入 JDBC 类或包括 JDBC 类的包
 - 加载 JDBC 驱动程序
 - 建立与数据库的连接
 - 执行 SQL 语句，与数据库交互
 - 关闭连接

第三章： JDBC 高级应用

目标：

本章旨在向学员介绍JDBC 的高级应用，通过本课的学习，学员应该掌握如下知识：

1) 掌握PreparedStatement 类的编码使用

学时：1学时

教学方法：讲授ppt
+上机练习

3.1 PreparedStatement 类

- PreparedStatement 类是 Statement 类的子类，允许使用不同的参数多次执行同样的 SQL 语句。
- Connection 类提供创建 PreparedStatement 对象的方法，可指定 SQL 语句：

```
PreparedStatement prepareStatement(String sql)  
throws SQLException
```

PreparedStatement 类示例

```
PreparedStatement pstmt = con.prepareStatement  
    ("INSERT INTO EMP VALUES(?,?)");  
pstmt.setInt(1, 99);  
pstmt.setString(2, "Tom");  
int count = pstmt.executeUpdate( );
```

3.2 setXxx 方法

Method	SQL Type
setASCIIStream	LONGVARCHAR produced by an ASCII stream
setBigDecimal	NUMERIC
setBinaryStream	LONGVARBINARY
setBoolean	BIT
setByte	TINYINT
setBytes	VARBINARY or LONGVARBINARY (depending on the size relative to the limits on VARBINARY)
setDate	DATE
setDouble	DOUBLE

setXxx 方法 (续)

Method	SQL Type
setFloat	FLOAT
setInt	INTEGER
setLong	BIGINT
setNull	NULL
setObject	The given object that is converted to the target SQL type before being sent
setShort	SMALLINT
setString	VARCHAR or LONGVARCHAR (depending on the size relative to the driver's limits on VARCHAR)
setTime	TIME
setTimestamp	TIMESTAMP

示例

- 修改 JdbcOracleTest 类代码，尝试使用 PreparedStatement 类发送 SQL 命令，实现对数据库的操作。

3.3 PreparedStatement 类的优点

- PreparedStatement 类的优点：
 - 可动态设置参数
 - 增加了预编译功能
 - 提高执行速度

3.4 事务处理

• 事务的概念

- ✓ **事务(Transaction):**是由相关SQL操作构成的一个完整的操作单元，该单元将被作为一个整体进行处理。
 - 执行事务的结果要不全部将数据所要执行的操作完成，要不全部数据都不修改
- ✓ **事务的4个属性:**
 - 原子性 (Atomicity)
 - 一致性 (Consistency)
 - 隔离性 (Isolation)
 - 持久性 (Durability)

- **JDBC中的事务处理**

- ✓ 取消自动提交模式
 - 即`setAutoCommit(false)`
- ✓ 当操作完成后，调用`commit()`方法提交对数据的修改
- ✓ 出错误时，调用`rollback()`放弃所做的修改

示例:

```
• try {  
    • Statement stmt=con.createStatement();  
    • //不自动提交事务  
    • con.setAutoCommit(false);  
    • //可以正确执行SQL  
    • stmt.executeUpdate("INSERT INTO EMP_MASTER VALUES  
    ('12','张三','学生','5000','3','30','F')");  
    • //不能正确执行的SQL  
    • stmt.executeUpdate("INSERT INTO EMP_MASTER " +  
    "VALUES ('哈哈','李四','学生','5000','3','30','F')");  
    • //执行完毕以后一定要手动提交  
    • con.commit();  
    } catch (SQLException e) {  
    • //回滚操作  
    • System.out.println("rollback");  
    • con.rollback();  
    • }  
}
```

3.5 小结

- PreparedStatement 类
- JDBC事务处理