



JDBC-1

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

INSERT / UPDATE / DELETE ; SELECT



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

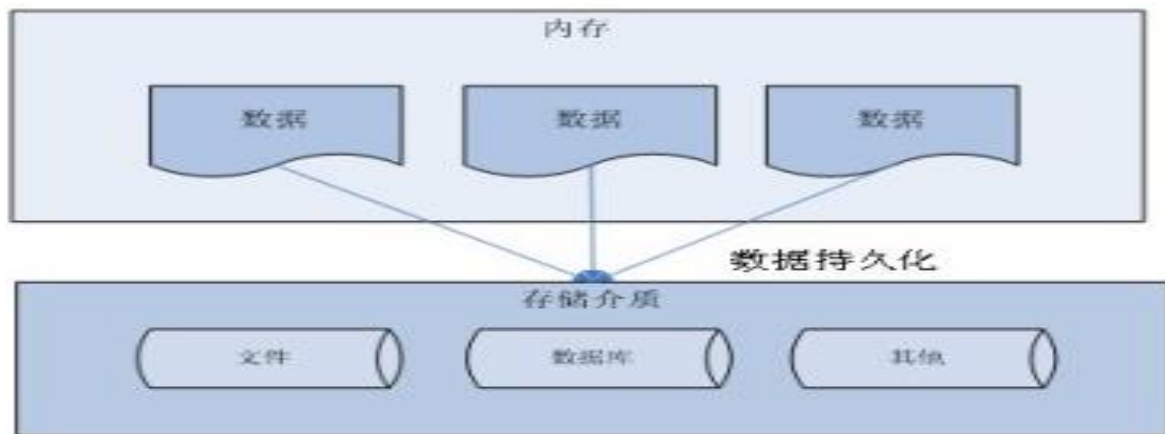


1-JDBC概述



序言：数据持久化

- 持久化(persistence): 把数据保存到可掉电式存储设备中以便之后使用。大多数情况下，特别是企业级应用，数据持久化意味着将内存中的数据保存到硬盘上加以“固化”，而持久化的实现过程大多通过各种关系数据库来完成。
- 持久化的主要应用是将内存中的数据存储在关系型数据库中，当然也可以存储在磁盘文件、XML数据文件中。





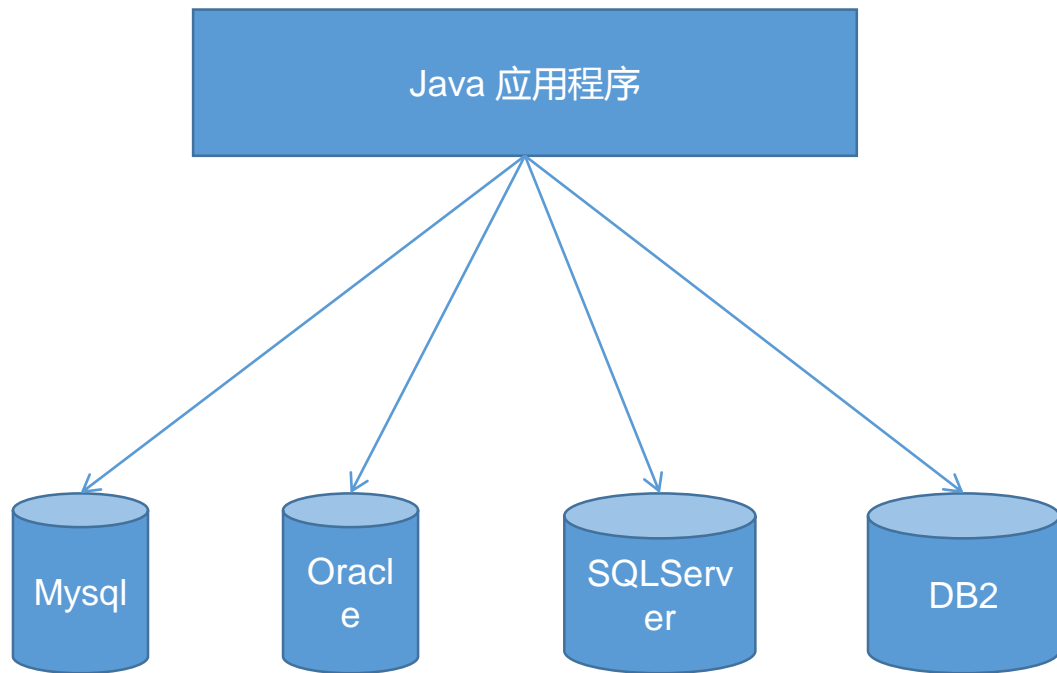
Java 中的数据存储技术

- 在Java中，数据库存取技术可分为如下几类：
 - JDBC**直接访问数据库
 - JDO技术
 - 第三方O/R工具，如Hibernate, mybatis 等
- JDBC是java访问数据库的基石，JDO, Hibernate等只是更好的封装了JDBC。

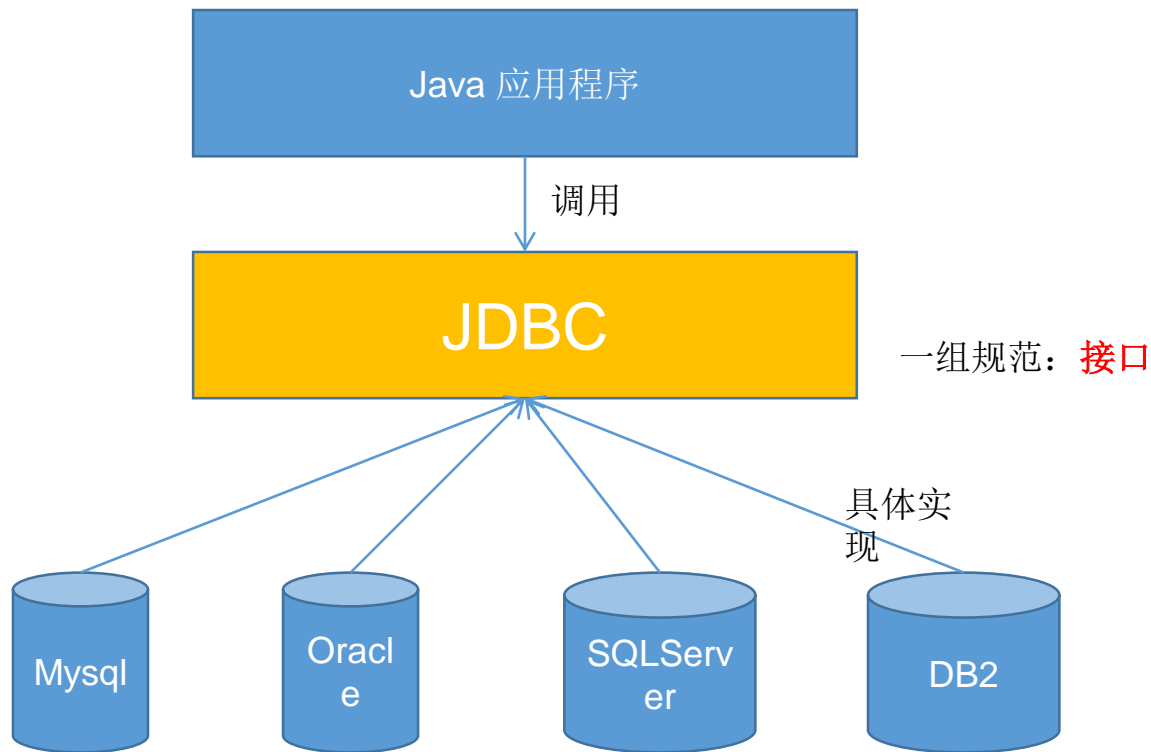


JDBC基础

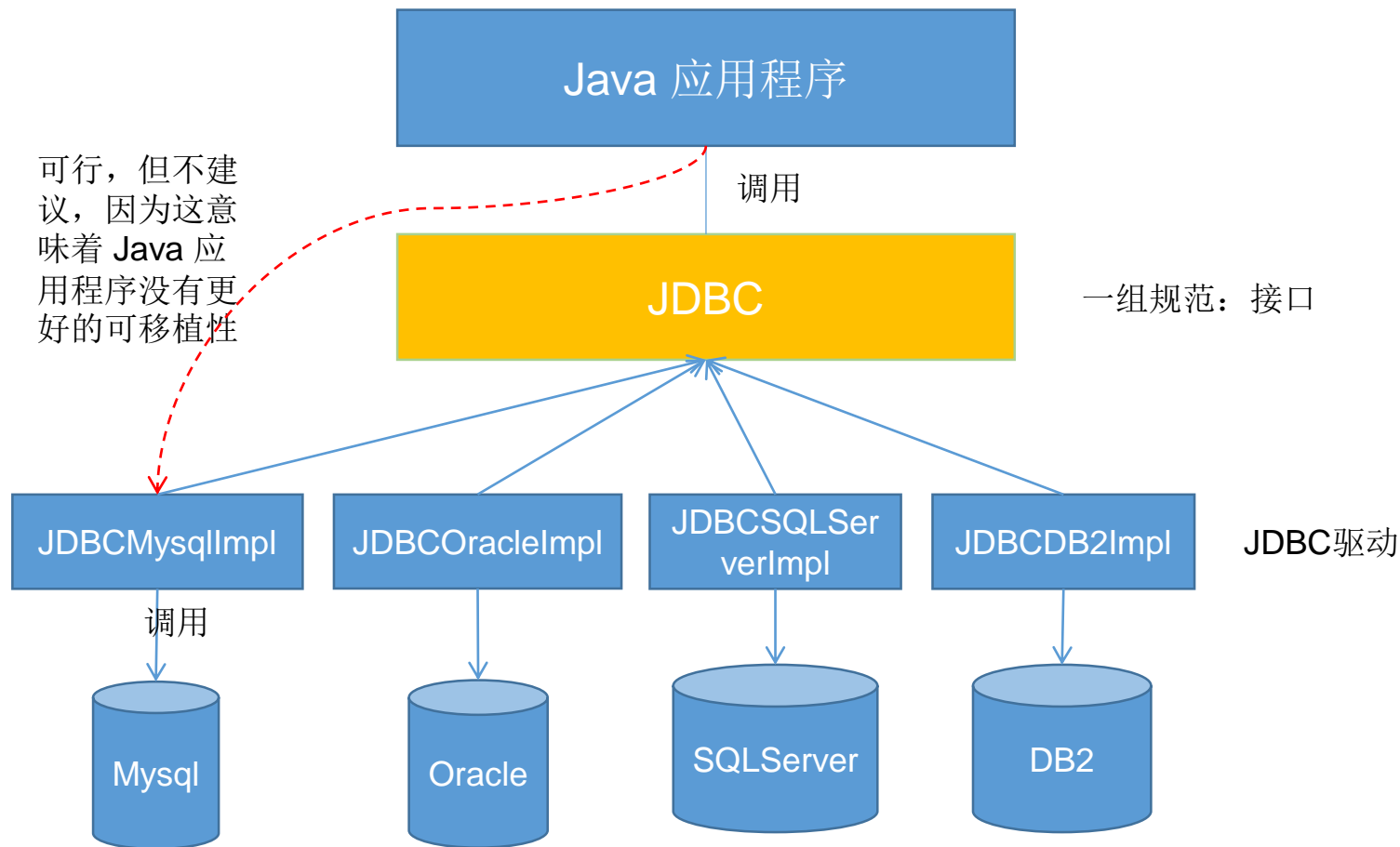
- JDBC(Java Database Connectivity)是一个**独立于特定数据库管理系统、通用的SQL数据库存取和操作的公共接口**（一组API），定义了用来访问数据库的标准Java类库，（java.sql,javax.sql）使用这个类库可以以一种**标准**的方法、方便地访问数据库资源
- JDBC为访问不同的数据库提供了一种**统一的途径**，为开发者屏蔽了一些细节问题。
- JDBC的目标是使Java程序员使用JDBC可以连接任何**提供了JDBC驱动程序**的数据库系统，这样就使得程序员无需对特定的数据库系统的特点有过多的了解，从而大大简化和加快了开发过程。



我们认为的连接应该这样



真实的连接是这样



综述

让天下没有难学的技术



JDBC体系结构

●JDBC接口（API）包括两个层次：

- 面向应用的**API**：Java API，抽象接口，供应用程序开发人员使用（连接数据库，执行SQL语句，获得结果）。
- 面向数据库的**API**：Java Driver API，供开发商开发数据库驱动程序用。

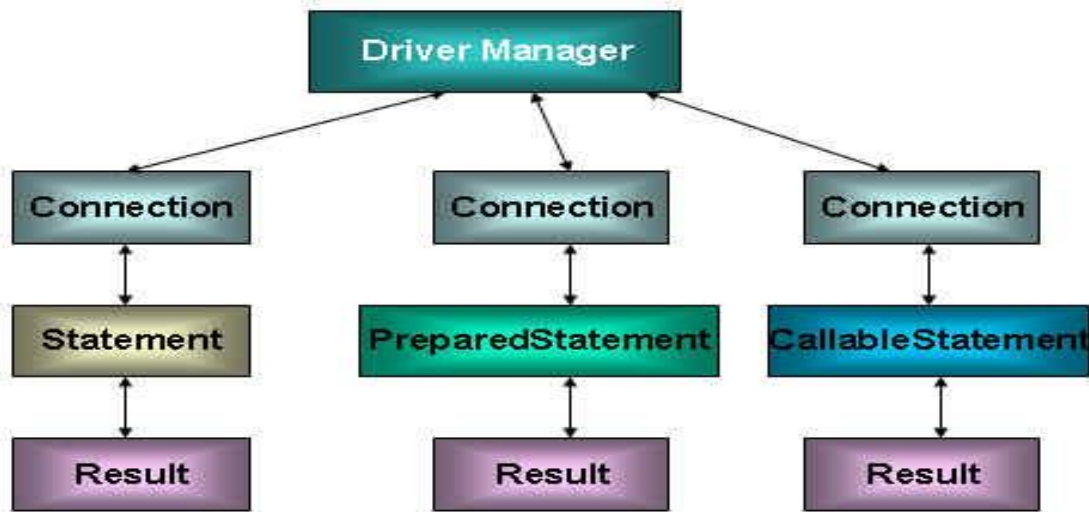
JDBC是sun公司提供一套用于数据库操作的接口，java程序员只需要面向这套接口编程即可。
不同的数据库厂商，需要针对这套接口，提供不同实现。不同的实现的集合，即为不同数据库的驱动。

—————面向接口编程



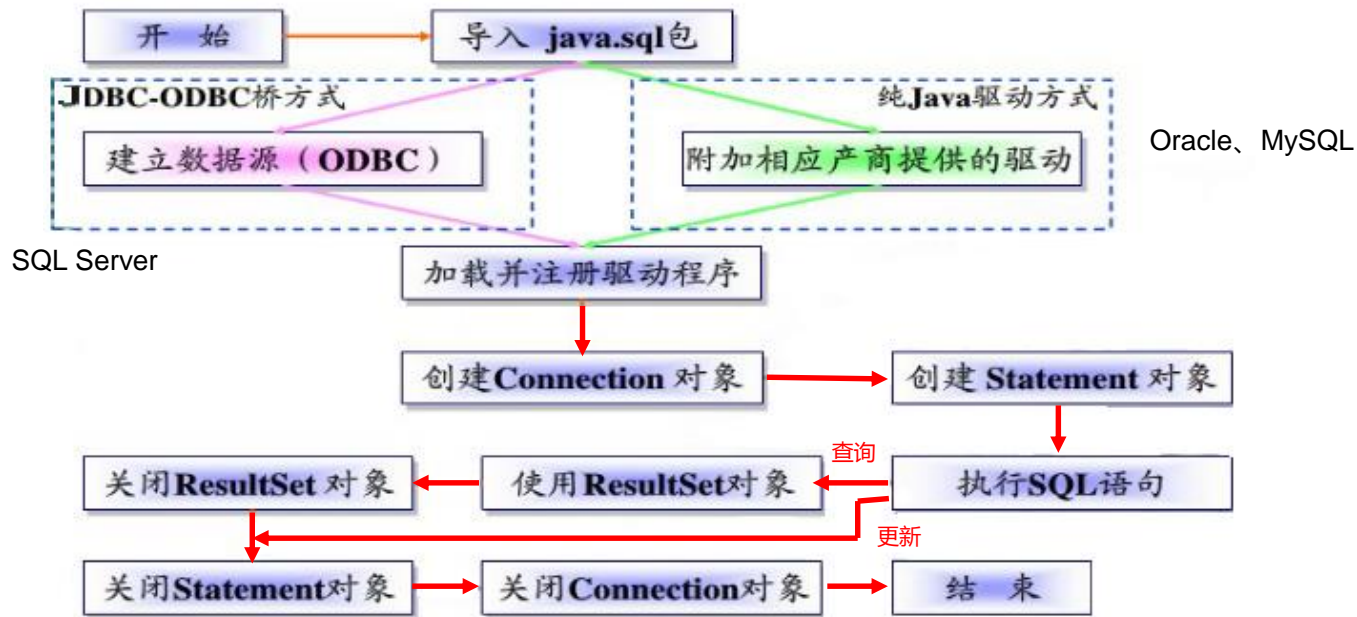
JDBC API

- JDBC API 是一系列的接口，它使得应用程序能够进行数据库联接，执行 SQL 语句，并且得到返回结果。





JDBC程序访问数据库步骤



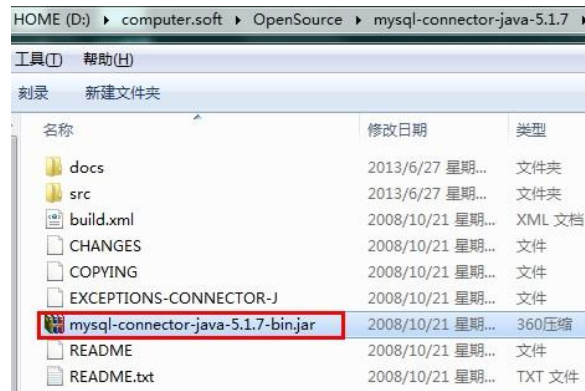
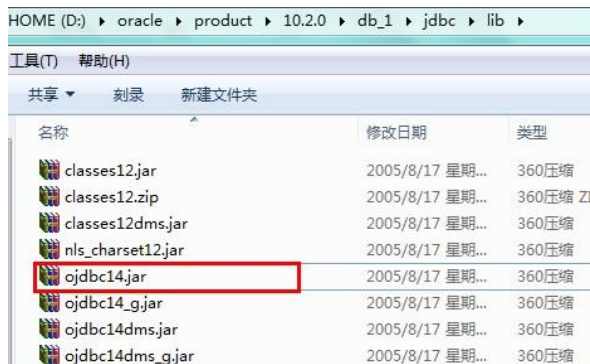


2-获取数据库连接



Driver 接口

- `java.sql.Driver` 接口是所有 JDBC 驱动程序需要实现的接口。这个接口是提供给数据库厂商使用的，不同数据库厂商提供不同的实现
- 在程序中不需要直接去访问实现了 `Driver` 接口的类，而是由驱动程序管理器类(`java.sql.DriverManager`)去调用这些 `Driver` 实现
 - Oracle 的驱动: `oracle.jdbc.driver.OracleDriver`
 - mySql 的驱动: `com.mysql.jdbc.Driver`





加载与注册 JDBC 驱动

- 方式一：加载 JDBC 驱动需调用 Class 类的静态方法 `forName()`，向其传递要加载的 JDBC 驱动类名

- **`Class.forName("com.mysql.jdbc.Driver");`**

- 方式二：DriverManager 类是驱动程序管理器类，负责管理驱动程序

- **`DriverManager.registerDriver(com.mysql.jdbc.Driver);`**

- 通常不用显式调用 DriverManager 类的 `registerDriver()` 方法来注册驱动程序类的实例，因为 Driver 接口的驱动程序类都包含了静态代码块，在这个静态代码块中，会调用 `DriverManager.registerDriver()` 方法来注册自身的一个实例

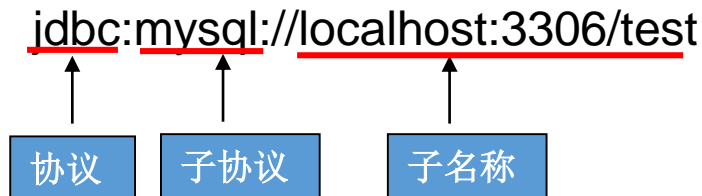


建立连接(Connection)

- 可以调用 DriverManager 类的 getConnection() 方法建立到数据库的连接
- User,password 可以用“属性名=属性值”方式告诉数据库;
- JDBC URL 用于标识一个被注册的驱动程序, 驱动程序管理器通过这个 URL 选择正确的驱动程序, 从而建立到数据库的连接。
- JDBC URL 的标准由三部分组成, 各部分间用冒号分隔。
 - **jdbc:子协议:子名称**
 - 协议: JDBC URL 中的协议总是 jdbc
 - 子协议: 子协议用于标识一个数据库驱动程序
 - 子名称: 一种标识数据库的方法。子名称可以依不同的子协议而变化, 用于子名称的目的是为了定位数据库提供足够的信息。包含主机名(对应服务端的ip地址), 端口号, 数据库名



几种常用数据库的JDBC URL



- 对于 Oracle 数据库连接，采用如下形式：
➤ **`jdbc:oracle:thin:@localhost:1521:atguigu`**
- 对于 SQLServer 数据库连接，采用如下形式：
➤ **`jdbc:microsoft:sqlserver//localhost:1433; DatabaseName=sid`**
- 对于 MYSQL 数据库连接，采用如下形式：
➤ **`jdbc:mysql://localhost:3306/atguigu`**



3-使用Statement操作数据表的弊端



访问数据库

- 数据库连接被用于向数据库服务器发送命令和 SQL 语句，在连接建立后，需要对数据库进行访问，执行 sql 语句
- 在 java.sql 包中有 3 个接口分别定义了对数据库的调用的不同方式：
 - Statement
 - ✓ PreparedStatement
 - CallableStatement



SQL 注入攻击

- SQL 注入是利用某些系统没有对用户输入的数据进行充分的检查，而在用户输入数据中注入非法的 SQL 语句段或命令(如：SELECT user, password FROM user_table WHERE user='a' OR 1 = ' AND password = ' OR '1' = '1')，从而利用系统的 SQL 引擎完成恶意行为的做法
- 对于 Java 而言，要防范 SQL 注入，只要用 PreparedStatement(从 Statement 扩展而来) 取代 Statement 就可以了



4-使用 **PreparedStatement**



PreparedStatement

- 可以通过调用 Connection 对象的 preparedStatement() 方法获取 PreparedStatement 对象
- PreparedStatement 接口是 Statement 的子接口，它表示一条预编译过的 SQL 语句
- PreparedStatement 对象所代表的 SQL 语句中的参数用问号(?)来表示，调用 PreparedStatement 对象的 setXxx() 方法来设置这些参数. setXxx() 方法有两个参数，第一个参数是要设置的 SQL 语句中的参数的索引(从 1 开始)，第二个是设置的 SQL 语句中的参数的值



PreparedStatement vs Statement

- 代码的可读性和可维护性.
- PreparedStatement 能最大可能提高性能:
 - DBServer会对**预编译**语句提供性能优化。因为预编译语句有可能被重复调用，所以语句在被DBServer的编译器编译后的执行代码被缓存下来，那么下次调用时只要是相同的预编译语句就不需要编译，只要将参数直接传入编译过的语句执行代码中就会得到执行。
 - 在statement语句中,即使是相同操作但因为数据内容不一样,所以整个语句本身不能匹配,没有缓存语句的意义.事实是没有数据库会对普通语句编译后的执行代码缓存.这样每执行一次都要对传入的语句编译一次.
 - (语法检查，语义检查，翻译成二进制命令，缓存)
- PreparedStatement 可以防止 SQL 注入



数据类型转换表

| java类型 | SQL类型 |
|--------------------|--------------------------|
| boolean | BIT |
| byte | TINYINT |
| short | SMALLINT |
| int | INTEGER |
| long | BIGINT |
| String | CHAR,VARCHAR,LONGVARCHAR |
| byte array | BINARY , VAR BINARY |
| java.sql.Date | DATE |
| java.sql.Time | TIME |
| java.sql.Timestamp | TIMESTAMP |



连接数据库、操作表的步骤

- 注册驱动 (只做一次)
- 建立连接(Connection)
- 创建执行SQL的语句 (Statement)
- 执行语句
- 处理执行结果(ResultSet)
- 释放资源

```
Connection conn = null;
Statement st=null;
ResultSet rs = null;
try {
    //获得Connection
    //创建Statement
    //处理查询结果ResultSet
} catch (Exception e){
    e.printStackTrace();
} finally {
    //释放资源ResultSet,
    // Statement,Connection
}
```



释放资源

- 释放ResultSet, Statement, Connection。
- 数据库连接（Connection）是非常稀有的资源，用完后必须马上释放，如果Connection不能及时正确的关闭将导致系统宕机。Connection的使用原则是**尽量晚创建，尽量早的释放。**



ResultSet

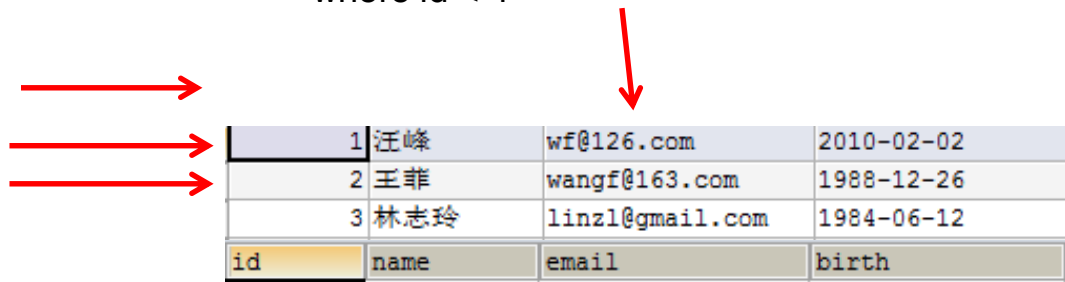
- 通过调用 `PreparedStatement` 对象的 `executeQuery()` 方法创建该对象
- `ResultSet` 对象以逻辑表格的形式封装了执行数据库操作的结果集，`ResultSet` 接口由数据库厂商实现
- `ResultSet` 对象维护了一个指向当前数据行的游标，初始的时候，游标在第一行之前，可以通过 `ResultSet` 对象的 `next()` 方法移动到下一行
- `ResultSet` 接口的常用方法：
 - `boolean next()`
 - `getString()`
 - ...



```
select id,email,name,birth from customers  
where id < 4
```

ResultSet rs

rs.next()



| | | | |
|----|------|-----------------|------------|
| 1 | 汪峰 | wf@126.com | 2010-02-02 |
| 2 | 王菲 | wangf@163.com | 1988-12-26 |
| 3 | 林志玲 | linz1@gmail.com | 1984-06-12 |
| id | name | email | birth |

```
rs.getInt(1) rs.getString(2) rs.getString(3) rs.getDate(4)
```



select id,email,name,birth from customers
where id < 4

ResultSet rs

rs.next()

| | | | |
|----|------|-----------------|------------|
| 1 | 汪峰 | wf@126.com | 2010-02-02 |
| 2 | 王菲 | wangf@163.com | 1988-12-26 |
| 3 | 林志玲 | linzl@gmail.com | 1984-06-12 |
| id | name | email | birth |

rs.getInt(1)

rs.getString(2)

rs.getString(3)

rs.getDate(4)

```
public class Customer {  
    private int id;  
    private String name;  
    private String email;  
    private Date birth;  
}
```

ORM:Object Relation Mapping

表 与 类 对应

表的一行数据 与 类的一个对象对应

表的一列 与 类的一个属性对应



SELECT id, name, age, birth FROM customer_table

初始状态: 指向第一条记录的前面

next(): 若返回 true, 就向下移动一行

| | | | |
|----|-----------|-----|------------|
| 2 | ChenXiang | 25 | 1991-12-12 |
| 3 | Arose | 21 | 1990-12-13 |
| 4 | Mike | 26 | (NULL) |
| 5 | Jinpeng | 15 | 1990-11-11 |
| id | name | age | birth |

1. 数组: `new Object[]{4, "Mike", 26, null}`
2. **Customer**: 一条记录对应一个对象
 1. **id**
 2. **name**
 3. **age**
 4. **birth**

`getInt(1)`

`getString(2)`

`getInt(3)`

`getDate(4)`



关于ResultSet的说明

1. 查询需要调用Prepared Statement 的 `executeQuery()` 方法, 查询结果是一个 `ResultSet` 对象
2. 关于 `ResultSet`: 代表结果集
 - `ResultSet`: 结果集. 封装了使用 `JDBC` 进行查询的结果.
 - 调用 `PreparedStatement` 对象的 `executeQuery()` 可以得到结果集.
 - `ResultSet` 返回的实际上就是一张数据表. 有一个指针指向数据表的第一条记录的前面.
3. 可以调用 `next()` 方法检测下一行是否有效. 若有效该方法返回 `true`, 且指针下移. 相当于 `Iterator` 对象的 `hasNext()` 和 `next()` 方法的结合体
4. 当指针指向一行时, 可以通过调用 `getXxx(int index)` 或 `getXxx(int columnName)` 获取每一列的值.
 - 例如: `getInt(1)`, `getString("name")`
5. `ResultSet` 当然也需要进行关闭.

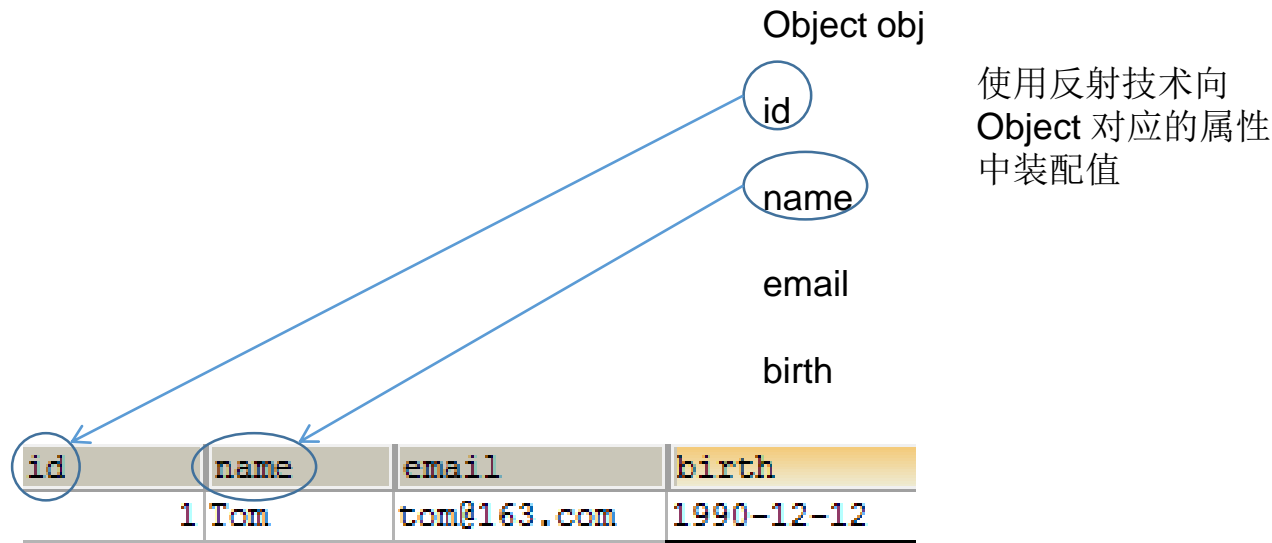


ResultSetMetaData 类

●可用于获取关于 **ResultSet** 对象中列的类型和属性信息的对象

●**ResultSetMetaData meta = rs.getMetaData();**

- **getColumnName(int column):** 获取指定列的名称
- **getColumnLabel(int column):** 获取指定列的别名
- **getColumnCount():** 返回当前 **ResultSet** 对象中的列数。
- **getColumnTypeName(int column):** 检索指定列的数据库特定的类型名称。
- **getColumnDisplaySize(int column):** 指示指定列的最大标准宽度，以字符为单位。
- **isNullable(int column):** 指示指定列中的值是否可以为 **null**。
- **isAutoIncrement(int column):** 指示是否自动为指定列进行编号，这样这些列仍然是只读的。



1. 得到结果集后, 如何知道该结果集中有哪些列? 列名是什么 ---- 需要使用一个描述 `ResultSet` 的对象, 即 `ResultSetMetaData`
2. `ResultSetMetaData`: 可以获取对应的 `ResultSet` 有多少列, 每一列的列名都是什么。
 1. 如何获取 `ResultSetMetaData`: 调用 `ResultSet` 的 `getMetaData()` 方法即可
 2. 获取 `ResultSet` 中有多少列: 调用 `ResultSetMetaData` 的 `getColumnCount()` 方法
 3. 获取 `ResultSet` 每一列的列的别名是什么。 `getColumnLabel()` 方法



String sql = “”;



得到结果集

ResultSet对象

| 1 | AA_ORDER | 1990-12-12 |
|---|----------|------------|
| 2 | BB_ORDER | 1990-12-13 |



ResultSetMetaData

| id | order_name | order_date |
|----|------------|------------|
| 1 | AA_ORDER | 1990-12-12 |
| 2 | BB_ORDER | 1990-12-13 |



| order_id | order_name | order_date |
|----------|------------|------------|
| 1 | AA | 2010-03-04 |
| 2 | BB | 2000-02-01 |
| 4 | GG | 1994-06-28 |
| 5 | CC | 1998-09-08 |
| 6 | DD | 1998-09-08 |
| 7 | MM | 1997-09-07 |

order表

```
select order_id id,order_name name,order_date date  
from order  
where order_id <5;
```

ResultSet

| id | name | date |
|----|------|------------|
| 1 | AA | 2010-03-04 |
| 2 | BB | 2000-02-01 |
| 4 | GG | 1994-06-28 |

```
public class Order {  
    private int id;  
    private String name;  
    private Date date;  
}
```

rs.getObject(1);

rsmd.getColumnLabel()

让天下没有难学的技术



| order_id | order_name | order_date |
|----------|------------|------------|
| 1 | AA | 2010-03-04 |
| 2 | BB | 2000-02-01 |

ORM思想

```
public class Order {  
    private int orderId;  
    private String orderName;  
    private Date orderDate;  
}
```

动态创建Order
的对象，并给
属性赋值

sql:select order_id orderId,order_name
orderName,order_date orderDate from 'order'
Where order_id < 4;

rs.next()

| orderId | orderName | orderDate |
|---------|-----------|------------|
| 1 | AA | 2010-03-04 |
| 2 | BB | 2000-02-01 |

1. 获取列数：rsmd.getColumnCount();
2. 获取列值：Object value = rs.getObject();
3. 获取列的别名：rsmd.getColumLabel()
4. T t = clazz.newInstance();



order表

| order_id | order_name | order_date |
|----------|------------|------------|
| 1 | AA | 2010-03-04 |
| 2 | BB | 2000-02-01 |
| 4 | MM | 1994-06-28 |

select order_id orderId,order_name orderName
from `order` where order_id = 1

结果集 : rs

rs.next()

| orderId | orderName | orderDate |
|---------|-----------|------------|
| 1 | AA | 2010-03-04 |

ORM的思想(object relational mapping)

- * 一个数据表 与 一个java类对应
- * 表中的一个列 与 java类的一个属性对应
- * 表中的一个行 与 java类的一个对象对应

列数 : rsmd.getColumnCount();
列别名 : rsmd.getColumnLabel(index);
列值 : rs.getObject(index);

```
public class Order {  
    private int orderId;  
    private String orderName;  
    private Date orderDate;  
}
```

T t = clazz.newInstance();

// 反射

Field field = clazz.getDeclaredField(columnLabel);
field.setAccessible(true);
field.set(t, columnValue);

t为Order类的对象



JDBC API 小结

●两种思想

➤ 面向接口编程的思想

➤ ORM思想

✓ sql是需要结合列名和表的属性名来写。注意起别名。

●两种技术

➤ JDBC元数据：ResultSetMetaData

➤ 通过反射，获取指定的属性，并赋值



练习1

1.从控制台向数据库的表**customers**中插入一条数据，表结构如下：

| cust_id | cust_name | email | birth | photo | |
|---------|-----------|-------------|------------|--------|----|
| 1 | 汪峰 | wf@126.com | 1990-12-12 | (NULL) | OK |
| 2 | 梁朝伟 | lcw@126.com | 1990-12-12 | (NULL) | OK |
| 4 | 刘德华 | LDH@126.com | 1990-12-12 | (NULL) | OK |
| (Auto) | (NULL) | (NULL) | (NULL) | (NULL) | OK |



练习2

1. 创立数据库表 **examstudent**，表结构如下：

| 字段名 | 说明 | 类型 |
|-------------|---------|-------------|
| FlowID | 流水号 | int(10) |
| Type | 四级 / 六级 | int(5) |
| IDCard | 身份证号码 | varchar(18) |
| ExamCard | 准考证号码 | varchar(15) |
| StudentName | 学生姓名 | varchar(20) |
| Location | 区域 | varchar(20) |
| Grade | 成绩 | int(10) |



练习2

2.向数据库中添加如下数据

| | | | | | | |
|---|---|--------------------|---------------------|-----|-----|----|
| 1 | 4 | 412824195263214584 | 20052316475400 0 | 张锋 | 郑州 | 85 |
| 2 | 4 | 222224195263214584 | 20052316475400 1 | 孙朋 | 大连 | 56 |
| 3 | 6 | 342824195263214584 | 20052316475400 2 | 刘明 | 沈阳 | 72 |
| 4 | 6 | 100824195263214584 | 20052316475400 3 | 赵虎 | 哈尔滨 | 95 |
| 5 | 4 | 454524195263214584 | 20052316475400 4 | 杨丽 | 北京 | 64 |
| 6 | 4 | 854524195263214584 | 20052316475400 5 | 王小红 | 太原 | 60 |



练习2

- 插入一个新的 student 信息

请输入考生的详细信息

Type:

IDCard:

ExamCard:

StudentName:

Location:

Grade:

信息录入成功!



练习2

3. 在 eclipse 中建立 java 程序：输入身份证号或准考证号可以查询到学生的基本信息。结果如下：

请选择您要输入的类型：

a: 准考证号

b: 身份证号

c

您的输入有误！请重新进入程序。

请选择您要输入的类型：

a: 准考证号

b: 身份证号

a

请输入准考证号：

200523164754004

=====查询结果=====

流水号： 5

四级/六级：4

身份证号： 454524195263214584

准考证号： 200523164754004

学生姓名： 杨丽

区域： 北京

成绩： 64

请选择您要输入的类型：

a: 准考证号

b: 身份证号

a

请输入准考证号：

20052316475

查无此人！请重新进入程序



练习2

完成学生信息的删除功能

请输入学生的考号:

45135324543632

查无此人，请重新输入！

请输入学生的考号:

6342634

删除成功！



向数据表中插入、读取大数据：**BLOB**字段



Oracle LOB介绍

- **LOB**，即**Large Objects**（大对象），是用来存储大量的二进制和文本数据的一种数据类型（一个LOB字段可存储可多达4GB的数据）。
- LOB 分为两种类型：内部LOB和外部LOB。
 - 内部LOB将数据以字节流的形式存储在数据库的内部。因而，内部LOB的许多操作都可以参与事务，也可以像处理普通数据一样对其进行备份和恢复操作。Oracle支持三种类型的内部LOB：
 - ✓ BLOB（二进制数据）
 - ✓ CLOB（单字节字符数据）
 - ✓ NCLOB（多字节字符数据）
 - CLOB和NCLOB类型适用于存储超长的文本数据，**BLOB字段适用于存储大量的二进制数据，如图像、视频、音频，文件等。**
 - 目前只支持一种外部LOB类型，即BFILE类型。在数据库内，该类型仅存储数据在操作系统中的位置信息，而数据的实体以外部文件的形式存在于操作系统的文件系统中。因而，该类型所表示的数据是只读的，不参与事务。该类型可帮助用户管理大量的由外部程序访问的文件。



MySQL BLOB 类型

- MySQL中，BLOB是一个二进制大型对象，是一个可以存储大量数据的容器，它能容纳不同大小的数据。
- MySQL的四种BLOB类型(除了在存储的最大信息量上不同外，他们是等同的)

| 类型 | 大小(单位：字节) |
|------------|-----------|
| TinyBlob | 最大 255 |
| Blob | 最大 65K |
| MediumBlob | 最大 16M |
| LongBlob | 最大 4G |

- 实际使用中根据需存入的数据大小定义不同的BLOB类型。
需要注意的是：如果存储的文件过大，数据库的性能会下降。



步 骤

- 向数据表中插入大数据类型

```
String sql = "INSERT INTO customer(name, email, birth, photo) VALUES(?, ?, ?, ?)";  
conn = JDBCUtil.getConnection();  
ps = conn.prepareStatement(sql);  
ps.setString(1, "LDH");  
ps.setString(2, "LDH@163.com");  
ps.setDate(3, new Date(new java.util.Date().getTime()));
```

//填充 **Blob** 类型的数据

```
ps.setBlob(4, new FileInputStream("abcd.jpg"));
```

```
ps.executeUpdate();
```



步骤

●从数据表中读取大数据类型

```
String sql = "SELECT id, name, email, birth, photo FROM customer WHERE id = ?";
conn = getConnection();
ps = conn.prepareStatement(sql);
ps.setInt(1, 8);
rs = ps.executeQuery();
if(rs.next()){
    Integer id = rs.getInt(1);String name = rs.getString(2);
    String email = rs.getString(3);Date birth = rs.getDate(4);
    Customer cust = new Customer(id, name, email, birth);System.out.println(cust);
    Blob photo = rs.getBlob(5);
    InputStream is = photo.getBinaryStream();
    OutputStream os = new FileOutputStream("c.jpg");
    byte [] buffer = new byte[1024];
    int len = 0;
    while((len = is.read(buffer)) != -1){
        os.write(buffer, 0, len);
    }
}
```



使用PreparedStatement实现批量插入



批量处理JDBC语句提高处理速度

- 当需要成批插入或者更新记录时。可以采用Java的批量**更新**机制，这一机制允许多条语句一次性提交给数据库批量处理。通常情况下比单独提交处理更有效率
- JDBC的批量处理语句包括下面两个方法：
 - addBatch(String): 添加需要批量处理的SQL语句或是参数;
 - executeBatch(): 执行批量处理语句;
 - clearBatch():清空缓存的数据
- 通常我们会遇到两种批量执行SQL语句的情况：
 - 多条SQL语句的批量处理;
 - 一个SQL语句的批量传参;



一个SQL语句的批量传参

优化1:

使用PreparedStatement替代Statement

优化2:

- ① 使用 addBatch() / executeBatch() / clearBatch()
- ② ?rewriteBatchedStatements=true&useServerPrepStmts=false
- ③ 使用更新的mysql 驱动:
mysql-connector-java-5.1.37-bin.jar

优化3:

Connection 的 setAutoCommit(false) / commit()



尚硅谷

