就业班课程回顾：

1. XML

XML技术 (2天)

--》 写XML

--》 读XML

第二部分：JavaWeb开发

Servlet / JSP

相关接口：

HttpServletRequest/response/ServletContext/HttpSession…….

第三部分：数据库

MySQL数据库

第四部分：JDBC

JDBC技术: java数据库连接技术！

接口：

Connection: 连接对象

Statement: 执行命令对象: 把SQL语句发送到数据库执行

ResultSet: (在线式)结果集接口, 必须要保持与数据库的连接！

开发步骤：

1. 建项目，引入数据库驱动包
2. 加载驱动

Class.forName(..);

1. 获取连接对象
2. 创建执行sql语句的stmt对象; 写sql
3. 执行sql
   1. 更新 delete/insert/update
      1. executeUpdate();
   2. 查询 select
      1. executeQuery();
4. 关闭/异常

目标:

1. 预编译sql处理(防止sql注入)
2. 批处理
3. 插入数据，获取自增长值
4. 事务
5. Jdbc中大文本类型的处理
6. Jdbc综合练习

明天/后天：

Jdbc技术优化、DbUtils组件、分页、连接池

# 预编译sql处理(防止sql注入)

|  |
| --- |
| -- 创建数据库  CREATE DATABASE jdbc\_demo DEFAULT CHARACTER SET utf8;i  -- 创建表  USE jdbc\_demo;  CREATE TABLE admin(  id INT PRIMARY KEY AUTO\_INCREMENT,  userName VARCHAR(20),  pwd VARCHAR(20)  ) |

|--Statement 执行SQL命令

|-- CallableStatement, 执行存储过程

|-- PreparedStatement 预编译SQL语句执行

使用预编译SQL语句的命令对象，好处：

1. 避免了频繁sql拼接 (可以使用占位符)
2. 可以防止sql注入

登陆模块，

输入用户名，密码！

注意，

要避免用户输入的恶意密码！

|  |
| --- |
| **public** **class** App {    // 连接参数  //private String url = "jdbc:mysql://localhost:3306/jdbc\_demo";  **private** String url = "jdbc:mysql:///jdbc\_demo";  **private** String user = "root";  **private** String password = "root";    **private** Connection con;  **private** Statement stmt;  **private** PreparedStatement pstmt;  **private** ResultSet rs;    // 1. 没有使用防止sql注入的案例  @Test  **public** **void** testLogin() {    // 1.0 模拟登陆的用户名，密码  String userName = "tom";  //String pwd = "8881";  String pwd = " ' or 1=1 -- ";    // SQL语句  String sql = "select \* from admin where userName='"+userName+"' and pwd='"+pwd+"' ";  System.*out*.println(sql);  **try** {  // 1.1 加载驱动，创建连接  Class.*forName*("com.mysql.jdbc.Driver");  con = DriverManager.*getConnection*(url, user, password);  // 1.2 创建stmt对象  stmt = con.createStatement();  // 1.3 执行查询  rs = stmt.executeQuery(sql);  // 业务判断  **if** (rs.next()) {  System.*out*.println("登陆成功, 编号：" + rs.getInt("id"));  }  } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  // 1.4 关闭  **try** {  rs.close();  stmt.close();  con.close();  } **catch** (Exception e) {  e.printStackTrace();  }  }  }      // 2. 使用PreparedStatement, 防止sql注入  @Test  **public** **void** testLogin2() {    // 1.0 模拟登陆的用户名，密码  String userName = "tom";  //String pwd = "8881";  String pwd = " ' or 1=1 -- ";    // SQL语句  String sql = "select \* from admin where userName=? and pwd=? ";  **try** {  // 1.1 加载驱动，创建连接  Class.*forName*("com.mysql.jdbc.Driver");  con = DriverManager.*getConnection*(url, user, password);  // 1.2 创建pstmt对象  pstmt = con.prepareStatement(sql); // 对sql语句预编译  // 设置占位符值  pstmt.setString(1, userName);  pstmt.setString(2, pwd);    // 1.3 执行  rs = pstmt.executeQuery();  **if** (rs.next()) {  System.*out*.println("登陆成功，" + rs.getInt("id"));  }    } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  // 1.4 关闭  **try** {  rs.close();  pstmt.close();  con.close();  } **catch** (Exception e) {  e.printStackTrace();  }  }  }      } |

# 存储过程调用

|  |
| --- |
| -- 存储过程  -- 定义分隔符  DELIMITER $$  CREATE PROCEDURE proc\_login()  BEGIN  SELECT \* FROM admin;  END $$  -- 调用  CALL proc\_login; |
| **public** **class** App\_call {    // 全局参数  **private** Connection con;  **private** Statement stmt;  **private** PreparedStatement pstmt;  **private** CallableStatement cstmt; // 存储过程  **private** ResultSet rs;    // 程序中调用存储过程  @Test  **public** **void** testCall() **throws** Exception {    **try** {  //1 . 创建连接  con = JdbcUtil.*getConnection*();  //2. 创建执行存储过程的stmt对象  CallableStatement cstmt = con.prepareCall("CALL proc\_login");  //3. 执行（存储过程）  rs = cstmt.executeQuery();    // 遍历结果，测试  **if** (rs.next()) {  String name = rs.getString("userName");  String pwd = rs.getString("pwd");  // 测试  System.*out*.println(name + pwd);  }    } **catch** (Exception e) {  e.printStackTrace();  }  }  } |

# 批处理

要在相同的statement中使用addBatch

很多时候，需要批量执行sql语句！

需求：批量保存信息！

设计：

AdminDao

Public void save(List<Admin list){ // 目前用这种方式

// 循环

// 保存 (批量保存)

}

Public void save(Admin admin ){

// 循环

// 保存

}

技术：

|-- Statement

批处理相关方法

void addBatch(String sql) 添加批处理

void clearBatch() 清空批处理

int[] executeBatch() 执行批处理

实现：

Admin.java 实体类封装数据

AdminDao.java 封装所有的与数据库的操作

App.java 测试

|  |
| --- |
| **public** **class** Admin {  **private** String userName;  **private** String pwd; |
| **public** **class** App {  // 测试批处理操作  @Test  **public** **void** testBatch() **throws** Exception {    // 模拟数据  List<Admin> list = **new** ArrayList<Admin>();  **for** (**int** i=1; i<21; i++) {  Admin admin = **new** Admin();  admin.setUserName("Jack" + i);  admin.setPwd("888" + i);  list.add(admin);  }    // 保存  AdminDao dao = **new** AdminDao();  dao.save(list);  }  } |

|  |
| --- |
| // 封装所有的与数据库的操作  **public** **class** AdminDao {    // 全局参数  **private** Connection con;  **private** PreparedStatement pstmt;  **private** ResultSet rs;  // 批量保存管理员  **public** **void** save(List<Admin> list) {  // SQL  String sql = "INSERT INTO admin(userName,pwd) values(?,?)";    **try** {    // 获取连接  con = JdbcUtil.*getConnection*();  // 创建stmt  pstmt = con.prepareStatement(sql); // 【预编译SQL语句】    **for** (**int** i=0; i<list.size(); i++) {  Admin admin = list.get(i);  // 设置参数  pstmt.setString(1, admin.getUserName());  pstmt.setString(2, admin.getPwd());    // 添加批处理  pstmt.addBatch(); // 【不需要传入SQL】    // 测试：每5条执行一次批处理  **if** (i % 5 == 0) {  // 批量执行  pstmt.executeBatch();  // 清空批处理  pstmt.clearBatch();  }    }    // 批量执行  pstmt.executeBatch();  // 清空批处理  pstmt.clearBatch();    } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  JdbcUtil.*closeAll*(con, pstmt, rs);  }  }  } |

# 插入数据，获取自增长值

* **需求**：

李俊杰 18

张相 19

如何设计数据库？

编号 员工姓名 年龄 部门

01 李俊杰 18 开发部

02 张三 19 开发部’

思考：

如何减少数据冗余？

🡪 设置外键约束

所以，

编号 员工姓名 年龄 部门

01 李俊杰 18 1

02 张三 19 1

部门编号 部门名称

1 开发部

部门与员工，

一对多的关系

* **设计数据库：**

员工表 （外键表） 【员工表有一个外键字段，引用了部门表的主键】

部门表（主键表）

* **编码总体思路:**

保存员工及其对应的部门！

步骤：

1. 先保存部门
2. 再得到部门主键，再保存员工

开发具体步骤：

1. 设计javabean
2. 设计dao
3. 测试

|  |
| --- |
| 部门  CREATE TABLE dept(  deptId INT PRIMARY KEY AUTO\_INCREMENT,  deptName VARCHAR(20)  );  -- 员工  CREATE TABLE employee(  empId INT PRIMARY KEY AUTO\_INCREMENT,  empName VARCHAR(20),  dept\_id INT -- 外键字段  );  -- 给员工表添加外键约束  ALTER TABLE employee ADD CONSTRAINT FK\_employee\_dept\_deptId  FOREIGN KEY(dept\_id) REFERENCES dept(deptId) ; |

|  |
| --- |
| **public** **class** EmpDao {    **private** Connection con;  **private** PreparedStatement pstmt;  **private** ResultSet rs;  // 保存员工，同时保存关联的部门  **public** **void** save(Employee emp){    // 保存部门  String sql\_dept = "insert into dept(deptName) values(?)";  // 保存员工  String sql\_emp = "INSERT INTO employee (empName,dept\_id) VALUES (?,?)";  // 部门id  **int** deptId = 0;    **try** {  // 连接  con = JdbcUtil.*getConnection*();    /\*\*\*\*\*保存部门，获取自增长\*\*\*\*\*\*\*/  // 【一、需要指定返回自增长标记】  pstmt = con.prepareStatement(sql\_dept,Statement.*RETURN\_GENERATED\_KEYS*);  // 设置参数  pstmt.setString(1, emp.getDept().getDeptName());  // 执行  pstmt.executeUpdate();    // 【二、获取上面保存的部门子增长的主键】  rs = pstmt.getGeneratedKeys();  // 得到返回的自增长字段  **if** (rs.next()) {  deptId = rs.getInt(1);  }    /\*\*\*\*\*保存员工\*\*\*\*\*\*\*\*\*/  pstmt = con.prepareStatement(sql\_emp);  // 设置参数  pstmt.setString(1, emp.getEmpName());  pstmt.setInt(2, deptId);  pstmt.executeUpdate();    } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  JdbcUtil.*closeAll*(con, pstmt, rs);  }  }  } |

# 事务

## 基本概念：

事务使指一组最小逻辑操作单元，里面有多个操作组成。 组成事务的每一部分必须要同时提交成功，如果有一个操作失败，整个操作就回滚。

事务ACID特性

* **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。
* **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。
* **隔离性（Isolation）**事务的隔离性是多个用户并发访问数据库时，数据库为每一个用户开启的事务，不能被其他事务的操作数据所干扰，多个并发事务之间要相互隔离。
* **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，接下来即使数据库发生故障也不应该对其有任何影响

事务的特性：

原子性，是一个最小逻辑操作单元 !

一致性，事务过程中，数据处于一致状态。

持久性， 事务一旦提交成功，对数据的更改会反映到数据库中。

隔离性， 事务与事务之间是隔离的。

## 案例

**需求**： 张三给李四转账

**设计**： 账户表

**技术**：

|-- Connection

void setAutoCommit(boolean autoCommit) ; 设置事务是否自动提交

如果设置为false，表示手动提交事务。

void commit() (); 手动提交 事务

void rollback() ; 回滚（出现异常时候，所有已经执行成功的代码需要回退到事务开始前的状态。）

Savepoint setSavepoint(String name)

**代码：**

|  |
| --- |
| -- 账户表  CREATE TABLE account(  id INT PRIMARY KEY AUTO\_INCREMENT,  accountName VARCHAR(20),  money DOUBLE  );  -- 转账  UPDATE account SET money=money-1000 WHERE accountName='张三';  UPDATE account SET money=money+1000 WHERE accountName='李四'; |

|  |
| --- |
| **public** **class** AccountDao {  // 全局参数  **private** Connection con;  **private** PreparedStatement pstmt;  // 1. 转账，没有使用事务  **public** **void** trans1() {  String sql\_zs = "UPDATE account SET money=money-1000 WHERE accountName='张三';";  String sql\_ls = "UPDATE account SET money=money+1000 WHERE accountName='李四';";  **try** {  con = JdbcUtil.*getConnection*(); // 默认开启的隐士事务  con.setAutoCommit(**true**);  /\*\*\* 第一次执行SQL \*\*\*/  pstmt = con.prepareStatement(sql\_zs);  pstmt.executeUpdate();  /\*\*\* 第二次执行SQL \*\*\*/  pstmt = con.prepareStatement(sql\_ls);  pstmt.executeUpdate();  } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  JdbcUtil.*closeAll*(con, pstmt, **null**);  }  }  // 2. 转账，使用事务  **public** **void** trans2() {  String sql\_zs = "UPDATE account SET money=money-1000 WHERE accountName='张三';";  String sql\_ls = "UPDATE1 account SET money=money+1000 WHERE accountName='李四';";  **try** {  con = JdbcUtil.*getConnection*(); // 默认开启的隐士事务  // 一、设置事务为手动提交  con.setAutoCommit(**false**);  /\*\*\* 第一次执行SQL \*\*\*/  pstmt = con.prepareStatement(sql\_zs);  pstmt.executeUpdate();  /\*\*\* 第二次执行SQL \*\*\*/  pstmt = con.prepareStatement(sql\_ls);  pstmt.executeUpdate();  } **catch** (Exception e) {  **try** {  // 二、 出现异常，需要回滚事务  con.rollback();  } **catch** (SQLException e1) {  }  e.printStackTrace();  } **finally** {  **try** {  // 三、所有的操作执行成功, 提交事务  con.commit();  JdbcUtil.*closeAll*(con, pstmt, **null**);  } **catch** (SQLException e) {  }  }  }  // 3. 转账，使用事务， 回滚到指定的代码段  **public** **void** trans() {  // 定义个标记  Savepoint sp = **null**;    // 第一次转账  String sql\_zs1 = "UPDATE account SET money=money-1000 WHERE accountName='张三';";  String sql\_ls1 = "UPDATE account SET money=money+1000 WHERE accountName='李四';";    // 第二次转账  String sql\_zs2 = "UPDATE account SET money=money-500 WHERE accountName='张三';";  String sql\_ls2 = "UPDATE1 account SET money=money+500 WHERE accountName='李四';";  **try** {  con = JdbcUtil.*getConnection*(); // 默认开启的隐士事务  con.setAutoCommit(**false**); // 设置事务手动提交  /\*\*\* 第一次转账 \*\*\*/  pstmt = con.prepareStatement(sql\_zs1);  pstmt.executeUpdate();  pstmt = con.prepareStatement(sql\_ls1);  pstmt.executeUpdate();    // 回滚到这个位置？  sp = con.setSavepoint();      /\*\*\* 第二次转账 \*\*\*/  pstmt = con.prepareStatement(sql\_zs2);  pstmt.executeUpdate();  pstmt = con.prepareStatement(sql\_ls2);  pstmt.executeUpdate();    } **catch** (Exception e) {  **try** {  // 回滚 (回滚到指定的代码段)  con.rollback(sp);  } **catch** (SQLException e1) {  }  e.printStackTrace();  } **finally** {  **try** {  // 提交  con.commit();  } **catch** (SQLException e) {  }  JdbcUtil.*closeAll*(con, pstmt, **null**);  }  }  } |

# Jdbc中大文本类型的处理

Oracle中大文本数据类型，

Clob 长文本类型 （MySQL中不支持，使用的是text）

Blob 二进制类型

MySQL数据库，

Text 长文本类型

Blob 二进制类型

需求： jdbc中操作长文本数据。

设计： 测试表

编码：

保存大文本数据类型

读取大文本数据类型

保存二进制数据

读取二进制数据

|  |
| --- |
| -- 测试大数据类型  CREATE TABLE test(  id INT PRIMARY KEY AUTO\_INCREMENT,  content LONGTEXT,  img LONGBLOB  ); |
| Text: |
| **public** **class** App\_text {    // 全局参数  **private** Connection con;  **private** Statement stmt;  **private** PreparedStatement pstmt;  **private** ResultSet rs;    @Test  // 1. 保存大文本数据类型 ( 写longtext)  **public** **void** testSaveText() {  String sql = "insert into test(content) values(?)";  **try** {  // 连接  con = JdbcUtil.*getConnection*();  // pstmt 对象  pstmt = con.prepareStatement(sql);  // 设置参数  // 先获取文件路径  String path = App\_text.**class**.getResource("tips.txt").getPath();  FileReader reader = **new** FileReader(**new** File(path));  pstmt.setCharacterStream(1, reader);    // 执行sql  pstmt.executeUpdate();    // 关闭  reader.close();  } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  JdbcUtil.*closeAll*(con, pstmt, **null**);  }  }    @Test  // 2. 读取大文本数据类型 ( 读longtext)  **public** **void** testGetAsText() {  String sql = "select \* from test;";  **try** {  // 连接  con = JdbcUtil.*getConnection*();  // pstmt 对象  pstmt = con.prepareStatement(sql);  // 读取  rs = pstmt.executeQuery();  **if** (rs.next()) {  // 获取长文本数据, 方式1:  //Reader r = rs.getCharacterStream("content");    // 获取长文本数据, 方式2:  System.*out*.print(rs.getString("content"));  }    } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  JdbcUtil.*closeAll*(con, pstmt, **null**);  }  }  } |

|  |
| --- |
| blob |
| **public** **class** App\_blob {    // 全局参数  **private** Connection con;  **private** Statement stmt;  **private** PreparedStatement pstmt;  **private** ResultSet rs;    @Test  // 1. 二进制数据类型 ( 写longblob)  **public** **void** testSaveText() {  String sql = "insert into test(img) values(?)";  **try** {  // 连接  con = JdbcUtil.*getConnection*();  // pstmt 对象  pstmt = con.prepareStatement(sql);  // 获取图片流  InputStream in = App\_text.**class**.getResourceAsStream("7.jpg");  pstmt.setBinaryStream(1, in);    // 执行保存图片  pstmt.execute();    // 关闭  in.close();    } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  JdbcUtil.*closeAll*(con, pstmt, **null**);  }  }    @Test  // 2. 读取大文本数据类型 ( 读longblob)  **public** **void** testGetAsText() {  String sql = "select img from test where id=2;";  **try** {  // 连接  con = JdbcUtil.*getConnection*();  // pstmt 对象  pstmt = con.prepareStatement(sql);  // 读取  rs = pstmt.executeQuery();  **if** (rs.next()) {  // 获取图片流  InputStream in = rs.getBinaryStream("img");  // 图片输出流  FileOutputStream out = **new** FileOutputStream(**new** File("c://1.jpg"));  **int** len = -1;  **byte** b[] = **new** **byte**[1024];  **while** ((len = in.read(b)) != -1) {  out.write(b, 0, len);  }  // 关闭  out.close();  in.close();  }    } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  JdbcUtil.*closeAll*(con, pstmt, **null**);  }  }  } |

# 综合案例

需求分析：

登陆、注册、注销；

登陆成功，

显示所有的员工

设计

数据库设计：

Admin, 存放所有的登陆用户

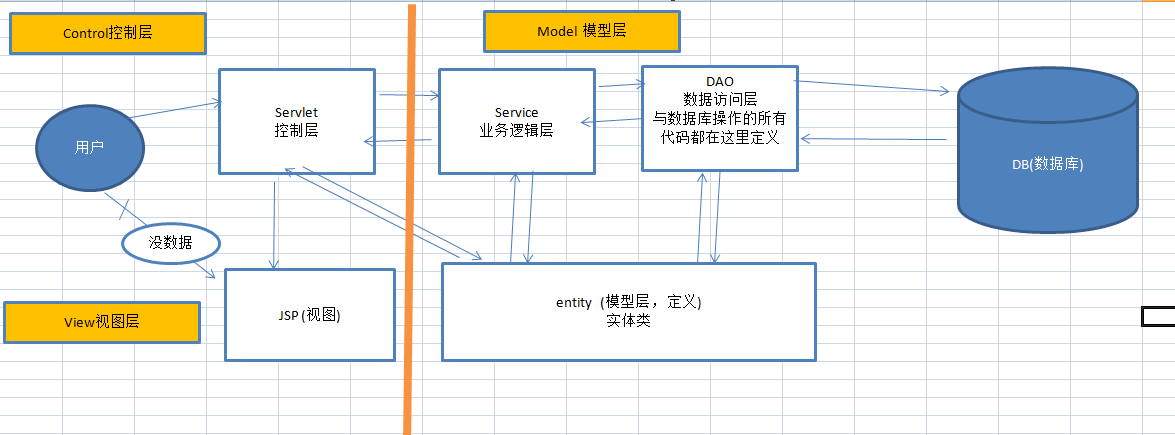
Employee, 存放所有的员工信息

系统设计

1. 系统结构

分层： 基于mvc模式的分层

1. 项目用到的公用组件、类 (了解)



编码

练习：

1. 案例
2. Jdbc增强知识点