Java实验报告

实验一 java 语言基本技能

实验目的

掌握java 语言的一些基本技能。

实验内容和过程

static域和方法的理解和使用 ,

this理解和使用

string不可变类的理解和使用

stringBuilder可变类的理解和使用

1）static方法

static方法一般称作静态方法，由于静态方法不依赖于任何对象就可以进行访问，因此对于静态方法来说，是没有this的，因为它不依附于任何对象，既然都没有对象，就谈不上this了。并且由于这个特性，在静态方法中不能访问类的非静态成员变量和非静态成员方法，因为非静态成员方法/变量都是必须依赖具体的对象才能够被调用。在静态方法中不能访问非静态成员方法和非静态成员变量，但是在非静态成员方法中是可以访问静态成员方法/变量的。

使用静态方法的实例：

public class Employee

{

public static int getNextId()

{

return nextId; //returns static field

}

}

//通过类名调用这个方法：

int n = Employee.getNextId();

2）this理解和使用

this代表它所在函数所属对象的引用

简单说：哪个对象在调用this所在的函数，this就代表哪个对象

一、方法调用模式

当函数被保存为一个对象的属性时，它就可称为这个对象的方法。当一个方法被调用时，this被绑定到这个对象上。如果调用表达式包含一个提取属性的动作（. 或 []），那么它被称为方法调用。例如：

var name = "window";

var obj = {

name: "kxy",

sayName: function() {

console.log(this.name);

}

};

obj.sayName(); //kxy

sayName函数作为对象obj的方法调用，所以函数体中的this就代表obj对象。

二、函数调用模式

当一个函数并非一个对象的属性时，那么它就是被当做函数来调用的。在此种模式下，this被绑定为全局对象，在浏览器环境下就是window对象。例如：

var name = "window";

function sayName() {

console.log(this.name);

}

sayName();

sayName以函数调用模式调用，所以函数体中的this代表window对象。

三、构造函数模式

如果在一个函数前面加上new关键字来调用，那么就会创建一个连接到该函数的prototype成员的新对象，同时，this会被绑定到这个新对象上。这种情况下，这个函数就可以成为此对象的构造函数。例如：

function Obj() {

this.name = "kxy";

}

var person = new Obj();

console.log(person.name); //kxy

Obj作为构造函数被调用，函数体内的this被绑定为新创建的对象person。

四、apply调用模式

在JS中，函数也是对象，所有函数对象都有两个方法：apply和call，这两个方法可以构建一个参数数组传递给调用函数，也允许改变this的值。例如：

var name = "window";

var person = {

name: "kxy"

};

function sayName() {

console.log(this.name);

}

sayName(); //window

sayName.apply(person); //kxy

sayName.apply(); //window

当以函数调用模式调用sayName时，this代表window；当用apply模式调用sayName，并给它传入的第一个参数为person时，this被绑定到person对象上。如果不给apply传入任何参数，则this代表window。

1. string不可变类的理解和使用

不可变对象就是创建后不可以改变的对象，

String s = "ABC";

s.toLowerCase();

.不可变对象可以提高String Pool的效率和安全性。如果你知道一个对象是不可变的，那么需要拷贝这个对象的内容时，就不用复制它的本身而只是复制它的地址，复制地址（通常一个指针的大小）需要很小的内存效率也很高。对于同时引用这个“ABC”的其他变量也不会造成影响。

不可变对象对于多线程是安全的，因为在多线程同时进行的情况下，一个可变对象的值很可能被其他进程改变，这样会造成不可预期的结果，而使用不可变对象就可以避免这种情况。

boolean connect(string s){

if (!isSecure(s)) {

throw new SecurityException();

}

// 如果在其他地方可以修改String,那么此处就会引起各种预料不到的问题/错误

causeProblem(s);

}

4）

stringBuilder可变类的理解和使用

String是不可变类，字符串都是常量，例如“ABC”会被存储在常量池中。对字符串进行任何更改操作都会产生新的String对象。而StringBuilder与StringBuffer是可变类，它们的字符串对象可以更改，对可变字符串的操作不会生成新的对象，即对同一个字符串对象操作。

StringBuilder sb = new StringBuilder(); //空字符串

StringBuilder sb1 = new StringBuilder("字符串"); //以String对象作为参数构建字符串

可变字符串的常用方法（增删改查）

append(data)　　　　　　　　//将给定数据作为字符串追加到可变字符串尾部

insert(offset, data) //将给定数据作为字符串追加到指定的偏移位置

delete(start, end); //删除start 到 end -1 的字符

replace(start, end, str) //替换start 到 end -1 的字符

reverse(); 　　　　　　 //将该字符串进行倒置

sb.setCharAt(index, ch);; //设置 index 处的字符为给定字符

StringBuilder sb = new StringBuilder(); //构建空的可变字符串

sb.append("String is changed");

System.out.println(sb); //原字符串对象内容被改变 String is changed

//在下标0处插入123

sb.insert(0, 123);

System.out.println(sb); //123String is changed

//删除 sb字符串的后面7位字符

sb.delete(sb.length() - 7, sb.length());

System.out.println(sb); //123String is

//替换前三位字符

sb.replace(0, 3, "!!!");

System.out.println(sb); //!!!String is

//翻转sb字符串

sb.reverse();

System.out.println(sb); // si gnirtS!!!

//设置某个下标的字符内容

sb.setCharAt(0,'A');

System.out.println(sb); //Asi gnirtS!!!

使用可变字符串的场景

1.将数组内容转成字符串

public static String toStringWithArray(int[] arr) {

StringBuilder sb = new StringBuilder("[");

for(int i = 0; i < arr.length; i++) {

if(i != arr.length -1) {

sb.append(arr[i] + ", ");

}else {

sb.append(arr[i] + "]");

}

}

return sb.toString();

}

如果不采用可变字符串，那么在拼接字符串过程中，会产生很多用不到的字符串对象，很浪费内存空间。

凡是需要大量拼接字符串的地方，都应该尽量使用可变字符串来完成，因为效率高，而且不占内存空间。

## 实验二 JDBC数据库编程

### 实验目的

学会使用JDBC存取典型关系数据库，比如MySql数据，对数据进行适当的处理。

### 实验内容和过程

驱动程序装载

数据库连接

数据库数据的存取

数据库数据的处理

实验完成情况：

组成JDBC的2个包（java.sql 、javax.sql）

（1）java.sql

类：DriverManager

接口：Connection Statement ResultSet PreparedStatement

CallableStatement（它用于调用存储过程）

（2）javax.sql

接口 DataSource

创建user表

create table user(

id int primary key auto\_increment,

username varchar(20) unique not null,

password varchar(20) not null,

email varchar(40) not null

);

INSERT INTO USER VALUES(NULL,'tom','123','tom@163.com');

INSERT INTO USER VALUES(NULL,'fox','123','fox@163.com');

编程从user表中读取数据，并打印在命令行窗口中。

（一）搭建实验环境

1、在MySQL中创建一个库，并创建user表和插入表的数据。

2、新建一个Java工程，并导入数据驱动

（二）编写程序，在程序中加载数据库驱动（称为注册驱动）

DriverManager.registerDriver(Driver driver);

（三）建立连接（Connection）

Connection conn = DriverManager.getConnection(url, user, pass);

（四）创建用于向数据库发送SQL的Statement对象，并发送SQL

Statement st = conn.createStatement();

ResultSet rs = st.executeQuery(sql);

（五）从代表结果集的ResultSet中取出数据，打印到命令行窗口

（六）断开与数据库的连接，并释放相关资源。

新建项目工程，创建cn.itcast.jdbc包 创建JdbcDemo1.java

package cn.itcast.jdbc;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import com.mysql.jdbc.Driver;

public class JdbcDemo1 {

public static void main(String[] args) throws SQLException {

// 1.注册驱动

DriverManager.registerDriver(new Driver());

// 2.获取连接对象

Connection con = DriverManager.getConnection(

"jdbc:mysql://localhost:3306/mydb1", "root", "root");

// 3.通过连接对象获取操作sql语句Statement

Statement st = con.createStatement();

// 4.操作sql语句

String sql = "select \* from user";

// 操作sql语句(select语句),会得到一个ResultSet结果集

ResultSet rs = st.executeQuery(sql);

// 5.遍历结果集

// boolean flag = rs.next(); // 向下移动，返回值为true，代表有下一条记录.

// int id = rs.getInt("id");

// String username=rs.getString("username");

// System.out.println(id);

// System.out.println(username);

while(rs.next()){

int id=rs.getInt("id");

String username=rs.getString("username");

String password=rs.getString("password");

String email=rs.getString("email");

System.out.println(id+" "+username+" "+password+" "+email);

}

//6.释放资源

rs.close();

st.close();

con.close();

}

}

## 实验三、四 Java web application

### 实验目的

了解和认识Java web application的涉及的核心技术、组成和开发过程

### 实验内容和过程

实现一个简单的学生信息管理：学生信息（学号、姓名、性别和年龄等）输入；学生信息的查询；

* 客户端设计与实现
  + HTML
  + CSS
  + Javascript
* 服务器端设计与实现
  + Servlet
  + Java
  + JDBC

实验实现结果

数据库

对数据库的封装代码：

1. 这里是定义数据库操作元素的结构，操作元素有三个属性，即deptno、dname、loc，分别对应学生信息的学号、姓名、家庭住址

package com.neusoft.dao;

//实体类

public class Dept {

private int deptno;//

private String dname;//

private String loc;//

private String value1;//

private String value2;//

public int getDeptno() {

return deptno;

}

public void setDeptno(int deptno) {

this.deptno = deptno;

}

public String getDname() {

return dname;

}

public void setDname(String dname) {

this.dname = dname;

}

public String getLoc() {

return loc;

}

public void setLoc(String loc) {

this.loc = loc;

}

public String getValue1() {

return value1;

}

public void setValue1(String value1) {

this.value1 = value1;

}

public String getValue2() {

return value2;

}

public void setValue2(String value2) {

this.value2 = value2;

}

}

（2）连接数据库：

DeUtils.java

package com.neusoft.dao;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class DbUtils {

private DbUtils(){}

private static final String url="jdbc:mysql://localhost:3306/smartstyle";;

private static final String user="root";

private static final String password="857289";

static

{

//1.创建驱动类对象

//new oracle.jdbc.driver.OracleDriver();

try {

//Class.forName("oracle.jdbc.driver.OracleDriver");

Class.forName("com.mysql.jdbc.Driver");

} catch (ClassNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

public static Connection getConnection()

{

Connection conn=null;

try {

conn=DriverManager.getConnection(url, user, password);

System.out.println("Ok");

} catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

return conn;

}

public static void closeConnection(Connection conn)

{

try {

if(conn!=null)

{

conn.close();

}

} catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

public static void closePreparedStatement(PreparedStatement pstmt)

{

try {

if(pstmt!=null)

{

pstmt.close();

}

} catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

public static void closeResultSet(ResultSet rs)

{

//4.关闭数据库

try {

if(rs!=null)

{

rs.close();

}

} catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

public static void beginTransaction(Connection conn) {

try {

conn.setAutoCommit(false);

} catch (SQLException ex) {

ex.printStackTrace();

}

}

public static void commit(Connection conn) {

try {

conn.commit();

conn.setAutoCommit(true);

} catch (SQLException ex) {

ex.printStackTrace();

}

}

public static void rollback(Connection conn) {

try {

conn.rollback();

conn.setAutoCommit(true);

} catch (SQLException ex) {

ex.printStackTrace();

}

}

public static void main(String[] ars)

{

getConnection();

}

}

（3）对数据库操作的封装：

这里的接口定义了对数据库操作的方法：

DeptDao.java

package com.neusoft.dao;

import java.util.List;

public interface DeptDao {

void insertDept(Dept dept);

void updateDept(Dept dept);

void deleteDept(int deptno);

List<Dept> getDepts();

Dept getDeptByDeptno(int deptno);

}

这里实现了对数据库操作的封装：

在建立数据库连接的基础上，通过向数据库发送要执行的SQL语句，达到操作数据库的目的。

DeptDaoImpl.java

package com.neusoft.dao;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

public class DeptDaoImpl implements DeptDao {

@Override

public void insertDept(Dept dept) {

// TODO Auto-generated method stub

Connection conn=null;

PreparedStatement pstmt=null;

try {

conn=DbUtils.getConnection();

//3.操作数据库

String sql="insert into dept(deptno,dname,loc) values(?,?,?)";

pstmt=conn.prepareStatement(sql);

pstmt.setInt(1, dept.getDeptno());

pstmt.setString(2, dept.getDname());

pstmt.setString(3,dept.getLoc());

pstmt.executeUpdate();

} catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}finally

{

//4.关闭数据库

DbUtils.closePreparedStatement(pstmt);

DbUtils.closeConnection(conn);

}

}

@Override

public void updateDept(Dept dept) {

// TODO Auto-generated method stub

Connection conn=null;

PreparedStatement pstmt=null;

try {

conn=DbUtils.getConnection();

String sql="update dept set dname=?,loc=? where deptno=?";

pstmt=conn.prepareStatement(sql);

pstmt.setString(1, dept.getDname());

pstmt.setString(2, dept.getLoc());

pstmt.setInt(3, dept.getDeptno());

pstmt.executeUpdate();

} catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}finally

{

//4.关闭数据库

DbUtils.closePreparedStatement(pstmt);

DbUtils.closeConnection(conn);

}

}

@Override

public void deleteDept(int deptno) {

// TODO Auto-generated method stub

Connection conn=null;

PreparedStatement pstmt=null;

try {

conn=DbUtils.getConnection();

String sql="delete from dept where deptno=?";

pstmt=conn.prepareStatement(sql);

pstmt.setInt(1, deptno);

pstmt.executeUpdate();

} catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}finally

{

//4.关闭数据库

DbUtils.closePreparedStatement(pstmt);

DbUtils.closeConnection(conn);

}

}

@Override

public List<Dept> getDepts() {

Connection conn=null;

PreparedStatement pstmt=null;

ResultSet rs=null;

List<Dept> depts=new ArrayList<Dept>();

try {

conn=DbUtils.getConnection();

//3.操作数据库

String sql="select \* from dept";

pstmt=conn.prepareStatement(sql);

rs=pstmt.executeQuery();

while(rs.next())

{

Dept dept=new Dept();

dept.setDeptno(rs.getInt("DEPTNO"));//

dept.setDname(rs.getString("DNAME"));

dept.setLoc(rs.getString("LOC"));

depts.add(dept);

}

}catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}finally

{

DbUtils.closeResultSet(rs);

DbUtils.closePreparedStatement(pstmt);

DbUtils.closeConnection(conn);

}

return depts;

}

@Override

public Dept getDeptByDeptno(int deptno) {

Connection conn=null;

PreparedStatement pstmt=null;

ResultSet rs=null;

Dept dept=new Dept();

try {

conn=DbUtils.getConnection();

//3.操作数据库

String sql="select \* from dept where deptno=?";

pstmt=conn.prepareStatement(sql);

pstmt.setInt(1, deptno);

rs=pstmt.executeQuery();

if(rs.next())

{

dept.setDeptno(rs.getInt("DEPTNO"));

dept.setDname(rs.getString("DNAME"));

dept.setLoc(rs.getString("LOC"));

}

}catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}finally

{

DbUtils.closeResultSet(rs);

DbUtils.closePreparedStatement(pstmt);

DbUtils.closeConnection(conn);

}

return dept;

}

}

（4）为了代码的简洁，再封装：

DeptService.java

package com.neusoft.service;

import java.util.List;

import com.neusoft.dao.Dept;

public interface DeptService {

void insertDept(Dept dept);

void updateDept(Dept dept);

void deleteDept(int deptno);

List<Dept> getDepts();

Dept getDeptByDeptno(int deptno);

}

DeptServiceImpl.java

package com.neusoft.service;

import java.util.List;

import com.neusoft.dao.Dept;

import com.neusoft.dao.DeptDao;

import com.neusoft.dao.DeptDaoImpl;

public class DeptServiceImpl implements DeptService {

@Override

public void insertDept(Dept dept) {

// TODO Auto-generated method stub

DeptDao deptDao=new DeptDaoImpl();

deptDao.insertDept(dept);

}

@Override

public void updateDept(Dept dept) {

// TODO Auto-generated method stub

DeptDao deptDao=new DeptDaoImpl();

deptDao.updateDept(dept);

}

@Override

public void deleteDept(int deptno) {

// TODO Auto-generated method stub

DeptDao deptDao=new DeptDaoImpl();

deptDao.deleteDept(deptno);

}

@Override

public List<Dept> getDepts() {

DeptDao deptDao=new DeptDaoImpl();

return deptDao.getDepts();

}

@Override

public Dept getDeptByDeptno(int deptno) {

// TODO Auto-generated method stub

DeptDao deptDao=new DeptDaoImpl();

return deptDao.getDeptByDeptno(deptno);

}

}

Web部分:

1、各个页面的JSP代码：

（1）index页面：

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>Insert title here</title>

</head>

<body>

<a href="DeptServlet?service=all">学生信息管理系统</a>

</body>

</html>

（2）主页面：

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8" import="java.util.List,com.neusoft.dao.Dept"%>

<%

String path = request.getContextPath();

String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";

%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>Insert title here</title>

</head>

<body>

<form action="<%=basePath %>dept/insert.jsp" method="post">

<table border="1" width="80%" align="center" cellspacing="0">

<caption>学生信息表</caption>

<tr><th>学生学号</th><th>学生姓名</th><th>家庭地址</th><th>操作</th><tr>

<%List<Dept> depts=(List<Dept>)request.getAttribute("depts");

if(depts==null)

{

%>

<tr><td colspan="3">没有符合条件的数据</td></tr>

<%}else{

for(Dept dept:depts)

{

%>

<tr><td><%=dept.getDeptno() %></td>

<td><%=dept.getDname() %></td>

<td><%=dept.getLoc() %></td>

<td><a href="DeptServlet?service=modify&deptno=<%=dept.getDeptno()%>">修改</a>

<a href="DeptServlet?service=delete&deptno=<%=dept.getDeptno() %>" >删除</a></td>

<tr>

<%}} %>

</table>

<input type="submit" value="添加" >

</form>

</body>

</html>

（3）修改页面：

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8" import="com.neusoft.dao.Dept"%>

<%

String path = request.getContextPath();

String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";

%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>Insert title here</title>

</head>

<body>

<form action="<%=basePath %>DeptServlet?service=update" method="post">

<center>

<h1>修改信息</h1>

<% Dept dept=(Dept)request.getAttribute("dept"); %>

学生学号：<input type="text" name="deptno" value="<%=dept.getDeptno() %>"><br>

学生姓名:<input type="text" name="dname" value="<%=dept.getDname() %>"><br>

家庭地址：<input type="text" name="loc" value="<%=dept.getLoc()%> "><br>

<input type="submit" value="修改信息">

</body>

</html>

2、服务器端：

对于客户端不同的HTTP请求对数据库进行不同的操作，并跳转页面。

package com.neusoft.controller;

import java.io.IOException;

import java.util.List;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import com.neusoft.dao.Dept;

import com.neusoft.service.DeptService;

import com.neusoft.service.DeptServiceImpl;

/\*\*

\* Servlet implementation class DeptServlet

\*/

@WebServlet("/DeptServlet")

public class DeptServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* Default constructor.

\*/

public DeptServlet() {

// TODO Auto-generated constructor stub

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

this.doPost(request, response);

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

request.setCharacterEncoding("UTF-8");

response.setCharacterEncoding("UTF-8");

String service=request.getParameter("service");

if("all".equals(service))

{

DeptService deptService=new DeptServiceImpl();

List<Dept> depts=deptService.getDepts();

request.setAttribute("depts", depts);

request.getRequestDispatcher("dept/main.jsp").forward(request, response);

}else if("add".equals(service))

{

//处理请求

int deptno=Integer.parseInt(request.getParameter("deptno"));

String dname=request.getParameter("dname");

String loc=request.getParameter("loc");

Dept dept=new Dept();

dept.setDeptno(deptno);

dept.setDname(dname);

dept.setLoc(loc);

//调用相应的业务逻辑

DeptService deptService=new DeptServiceImpl();

deptService.insertDept(dept);

//找到某个视图响应回去

request.getRequestDispatcher("DeptServlet?service=all").forward(request, response);

}else if("update".equals(service))

{

//处理请求

int deptno=Integer.parseInt(request.getParameter("deptno"));

String dname=request.getParameter("dname");

String loc=request.getParameter("loc");

Dept dept=new Dept();

dept.setDeptno(deptno);

dept.setDname(dname);

dept.setLoc(loc);

//调用相应的业务逻辑

DeptService deptService=new DeptServiceImpl();

deptService.updateDept(dept);

//找到某个视图响应回去

request.getRequestDispatcher("DeptServlet?service=all").forward(request, response);

}

else if("delete".equals(service))

{

int deptno=Integer.parseInt(request.getParameter("deptno"));

//调用相应的业务逻辑

DeptService deptService=new DeptServiceImpl();

deptService.deleteDept(deptno);

//找到某个视图响应回去

request.getRequestDispatcher("DeptServlet?service=all").forward(request, response);

}else if("modify".equals(service))

{

int deptno=Integer.parseInt(request.getParameter("deptno"));

//调用相应的业务逻辑

DeptService deptService=new DeptServiceImpl();

Dept dept= deptService.getDeptByDeptno(deptno);

request.setAttribute("dept", dept);

//找到某个视图响应回去

request.getRequestDispatcher("dept/update.jsp").forward(request, response);

}

}

}