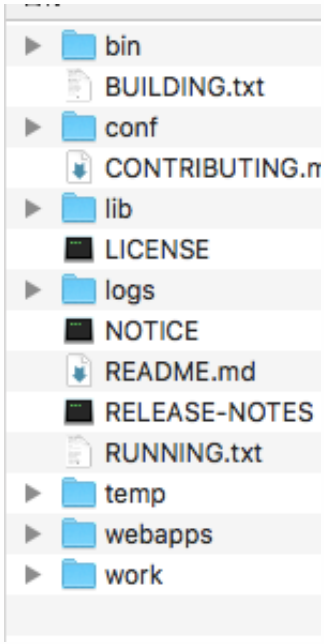
j2ee笔记

Tomcat原理



bin:服务器的启动目录（startup.sh,shutdown.sh）

conf:配置文件目录

lib：tomcat依赖的jar文件（里面的jar文件为全局jar）

logs:打印tomcat运行报错等信息

temp:临时文件的存放区域

webapps:可以执行的项目（tomcat默认的项目放在该目录下）

可以把项目直接丢到这里就可以了，用eclipse部署的访问不到这里是因为设置了虚拟路径

work:存放jsp转化的java文件以及class文件

jsp编译执行流程：当用户用浏览器第一次访问jsp时会编译jsp--》java--》jar存放于work目录，当用户再次访问时速度会比第一次快，因为jsp已经编译好了，他会访问已经编译好的jsp

web-inf中的文件一般不能访问，只有通过内部请求转发和重定向（安全）

配置虚拟路径（已经省略，实际用不到）

虚拟路径：默认是webapps

servlet只能在服务器上才能运行。

给Tomcat设置虚拟路径的三种方法

1在server.xml中添加如下代码

在标签中添加：

docBase:为实际的物理路径，上面是绝对路径

一定要注意：

docBase:为项目的物理路径

path：虚拟路径

在mac中不要填“\”与windows相反

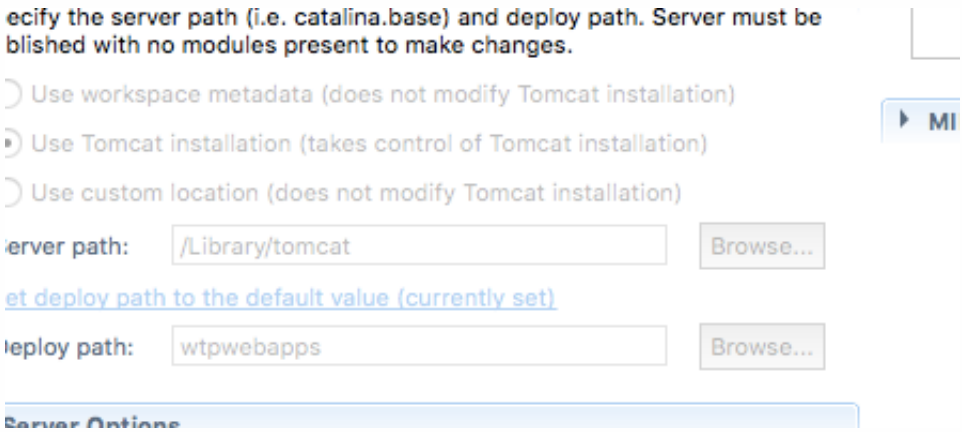
添加完后重启tomcat即可

2在conf中的

conf/Catalina/localhost 文件夹下新建test.xml 内容就是添加上述的
启动tomcat 即可

3直接把需要的项目放到webapps文件夹下（tomcat默认路径）

eclipse启动web项目原理也是如此部署到tomcat后会改掉项目的虚拟路径，所以部署完后不能直接访问tomcat下的webapps下的项目



选择第二个可以保证，eclipsse的tomacat和本地的tomcat一致

jsp页面元素

1脚本

scriptlet:

<%%>java代码，局部变量

<%!%>方法，全局变量

<%=%>表达式

```
<%System.out.println("Inside Async 1");out.write("lsdkjfl");%><%! String
name="hhh";public void ppp(){String ll="dkjflslioiweo";}%><!--用于全局变量和方法
--> <%="你的名字："+name %><!-- 主要用于输出类似于out.write() -->
```

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

pageEncoding="UTF-8"%>

pageEncoding：jsp文件的编码

language:jsp页面使用的语言

content-type：浏览器的解析jsp的编码

7:jsp的9大内置对象：（不需要new可以直接使用的）

out:输出对象，用于向客户端输出信息（继承 于java.io.writer）

request:储存客户端向服务器发送的请求信息

方法：

getparameter(String name)根据字段名key返回value

getparametervalues(String name)（chekbook）根据字段名key返回多个value

setCharacterEncoding()设置请求的字符编码，tomcat7以前是iso-8859-1，tomcat8是utf-8（一般用于post提交的设置）

getRequstDispatcher("b.jsp").forward(request,response)请求转发的方式跳转页面 a-》b.jsps

getservercontext():获取serverletcontext对象

response:服务端的响应对象

方法：

void addcookies(Cookie cookie):服务端向客户端增加的cookie对象

void sendRedirect(String location)throw exception :页面跳转的一种方式

setContentType(String type):设置服务端响应的编码（设置服务端的contentType）

重定向和转发的区别：

	重定向	转发
数据保留	不保留	保留
地址栏	新地址栏	原来的地址栏
跳转	产生一次新的请求	服务端内部

session：会话，不同浏览器第一次访问服务器就会产生一个session对象

session原理：会话客户端第一次请求服务端时，服务端会产生一个session对象（用于保存该客户的信息，）并且每一个session对象都会有唯一的一个sessionid（用于区分其他session），服务端会产生一个cookie（name="JSESSIONID"，value="服务端sessionid的值"）用于保存sessionid，然后服务端会响应给客户端，客户端会接收到cookie。与服务端session一一对应。（JSSIONID==>sissionid）

session第二次/n次服务端会用sessionid和JSSIONID匹配，如果能匹配，则能登陆。

方法：

String getid（）：获取sessionid

boolean isnew(): 是否为新用户

void invalidate():使session 无效

setattribute（）给session设置属性

getattribute（）获取session属性值

void setmaxinactiveinterval(秒):设置最大有效时间（非有效时间）

application： 全局对象

String getcontentextpath():获取当前对象的虚拟路径

<%out.write(application.getContextPath()//获取虚拟路径

); %>

getrealpath():获取当前对象的真实路径。

config：配置对象，当前服务器配置信息

page：当前jsp页面对象(相当于this)

exception：异常对象

pageContext：jsp的页面容器

****

cookie :服务器响应客户端的数据，由服务端response产生，客户端接收服务端的响应会在本地保存，相当于本地缓存

作用：跳高访问服务端的效率，但是安全性更差

cookie： 本质是键值对

javax.servlet.http.Cookie

构造方法 public Cookie(String name,String value)

void setMaxAge(int expiry):最大有效期

服务端准备cookie：

response.addCookie(Cookie cookie)

页面跳转重定向，转发：

客户端护墙cookie： request.getcookies()

cookie应用案例

客户端通过请求，服务器端response.addcookie(String name,String value)--》再发送一个响应，给客户端，客户端就能收到cookie了

<%!String name; %><%name=request.getParameter("uuuuu");Cookie cook=new Cookie("uuuuu",name) ;response.addCookie(cook); response.sendRedirect("NewFile.jsp");//发送一个响应给客户端，就可以把cookie保存到客户端了%>

session和cookie的区别：

session cookie

保存的位置 服务端 客户端

安全性 安全 不安全

保存的内容 object String

control+shift+t进入源文件

```
@RequestMapping("/test2")public void chongdingx(HttpServletResponse response
,HttpServletRequest request) throws IOException, ServletException {
System.out.println("来了");   // response.sendRedirect("index2.jsp");
//response.sendRedirect("WEB-INF/page/index2.jsp"); 转发无法访问web-inf下的jsp   //
response.sendRedirect("/WEB-INF/page/index2.jsp");
request.getRequestDispatcher("WEB-
INF/page/index2.jsp").forward(request,response);//重定向可以访问WEB-INF下的文件   //
return  null;}``用servlet api 跳转不会经过视图解析器（加前缀和后缀）
```

jsp内置对象的作用域

作用域的区别

1request：同一个请求有效，重定向无效

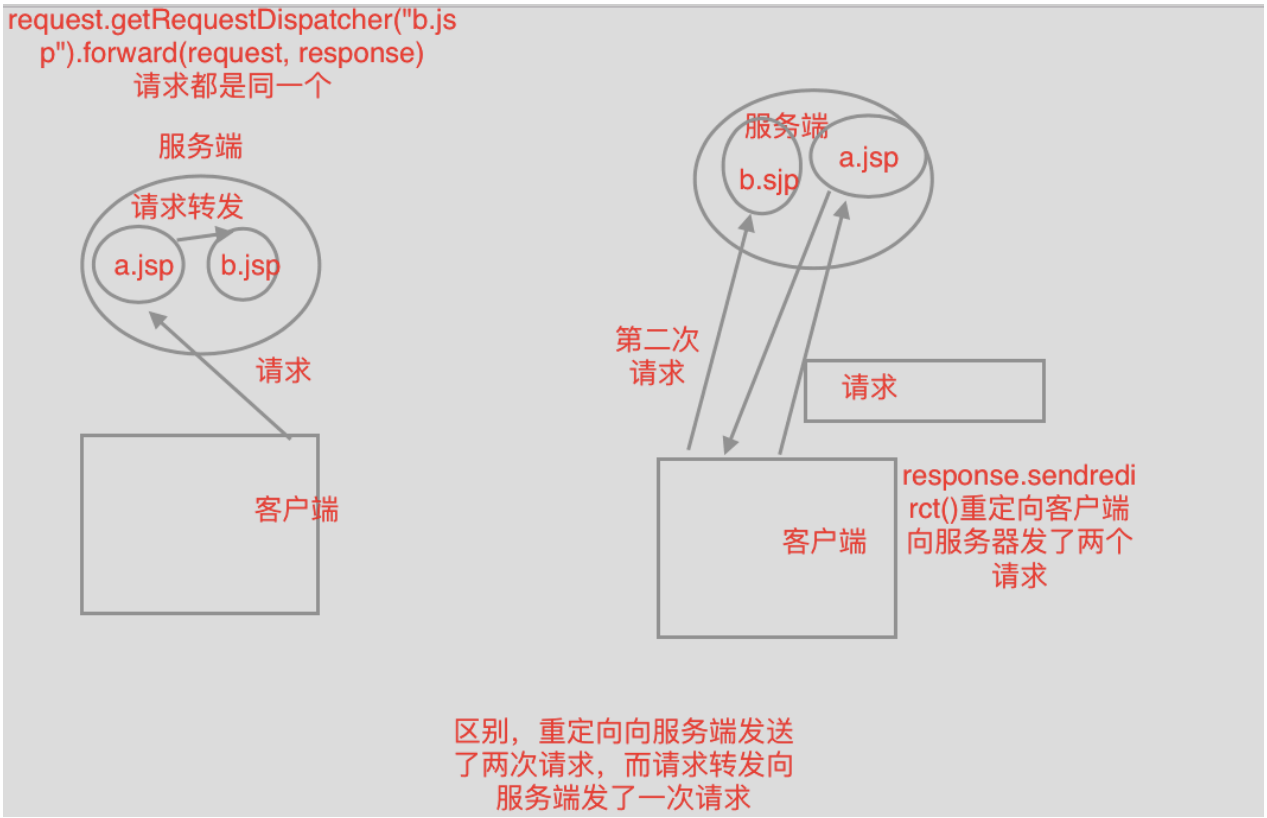
2application：全局对象

3session：同一次会话有效（从登陆服务器到退出过程之间都有效）

4pagecontext(也叫page对象)

都通过setattribute()赋值，getattribute取值

范围大小关系 4<1<3<2



处理乱码

在server.xml中加入

只有post方式有效。

jdbc

JDBC：java database connectivity 可以为Dbms提供统一的访问方式

jdbc api:接口， 方法，类

三件事 通过以下类/接口实现

建立数据库链接

发送sql语句

返回结果集

DriverManager:管理jdbc驱动

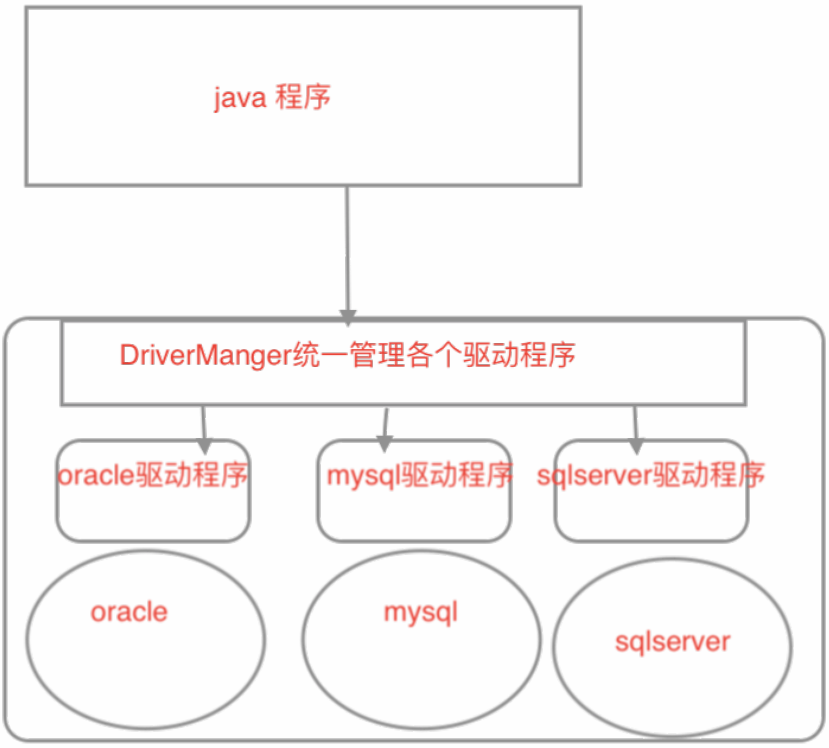
connection:连接

Statement（prepstatement）： 增删改查

Result:返回的结果集

mysql连接字符串

"jdbc:mysql://localhost:3306/数据库实例名"



将java 代码放入webproject

MVC设计模式

M:model 模型，一个功能，用javabean实现

v:view ： 视图用于人机交互

c： Controller：控制器：接收请求，将请求跳转到模型进行处理：模型处理完毕后再将请求的结果返回给请求处。可以用jsp实现。但是一般不这么干，用servlet实现。

servlet：

java类必须符合一定的规范：

a：必须实现httpservlet

b：必须重写其中的doget()或dopost()方法

doGet():处理get请求

doPost():处理post请求

Servlet要想使用，必须配置

servlet2.5 :web.xml

servlet3.0: @WebServlet(注解)

区别:servlet不需要配置web.xml但是要加@WebServlet（url）如图

过程：通过请求的路径匹配注解里的路径

Servlet01 com.baidu.dao.Servlet01 Servlet01 /Servlet01

servlet2.5

项目的根目录webcontent ,src（所以的构建路径）文件夹右键buildpath

web.xml中的"/"代表项目根路径(http://127.0.0.1:8080/项目名)，jsp中的“/”代表

http://127.0.0.1:8080/

servlet(生命周期)

5步

加载

初始化 init() 默认第一次访问servlet时调用（只调用一次）

也可以设置默认tomcat启动时添加

```
 test01
 test01
 com.baidu.dao.test01
 1
```

这是2.5的方式配置，1代表顺序，因为可能不止一个servlet

服务 service()是抽象方法，一般用子类的doget()和dopost()

销毁 destroy（）:关闭tomcat时调用(只调用一次)

卸载:

servlet api

servlet*继承关系*

ServletContext getServletContext（）： 获取servlet上下文对象 application

String getInitParameter(String name); 获取当前servlet 范围内，获取init参数

servlet25配置：

ds sdfdsf

ServletContext:

getContextpath

getattribute(),setarrtribute(),

getContextpath():绝对路径

getRealpath():虚拟路径

---->

getInitParameter(String name); 获取当前web容器 范围内，获取init参数

globalParam hhh

servlet25下获取初始化参数的方法

@Overridepublic void init() throws ServletException {// TODO Auto-generated method stubsuper.init();System.out.println("init方法被调用");String u= super.getInitParameter("ds");System.out.println("当前servlet的初始化值为： "+u);ServletContext servletcontext= super.getServletContext();String y= servletcontext.getInitParameter("globalParam");System.out.println("web容器的初始化参数为: "+y);}

//区别：作用域不同，一个是application的作用域，一个是servlet的作用域

（service（）是servlet的生命周期的一部分，当请求到某一个servlet时就会调用该方法）

servlet的父类httpservlet对service的重写：

先将ServletRequest转化为HttpServletRequest，ServletResponse 转为HttpServletResponse

再判断requsest的方法是那种类型（总共有七种请求类型），再调用具体的方法包括doget(),dopost()，一般情况servlet只要重写doget(),dopost()方法即可

三层架构

三层架构：

与mvc设计模式目标一致：为了解耦合，提高代码复用率

区别，二者对项目理解的角度不同

2

三层组成“

表示层（servler层）

-前台：对应mvc的view，用于和用户交互，界面的显示

--后台： 请求的分发

业务逻辑层

--接收表示层的请求，调用

组装数据访问层，逻辑性的操作（增删改查 删：查+删）service

数据访问层

-直接访问数据库的操作，原子性的操作（增删改查）dao

响应出现乱码解决方法：

response.setCharacterEncoding("utf-8");

response.setContentType("text/html; charset=UTF-8");

文件上传

enctype="multipart/form-data"

引用俩个jar包

csdn ***https://blog.csdn.net/weixin_45558236/article/details/105012139***

El表达式和jstl

EL：Expression Laguage 表达式语言

用于替代java代码

EL实例：

${requestScope.student.name}

${requestScope.student.hobbies[1]}

${域对象.域对象的属性.属性.级连属性}

[]操作符：可以匹配哪些特殊字符 [.,]

注意：域对象的对像属性一定要符合javabeans的写法规范

package com.baidu.entity; public class Student {private String name;private int age;public Student(String name, int age) {this.name = name;this.age = age;}public String getName() {return name;}public void setName(String name) {this.name = name;}public int getAge() {return age;}public void setAge(int age) {this.age = age;}}

获取map属性

Map<String, String>map=new HashMap<String, String>() ;

map.put("cn", "iiiii");

s.setMap(map);

request.setAttribute("student", s);

${requestScope.student.map.cn}

运算

${3>2}.${3 gt 2}

${3>2 || 2>3} ${3>2 or 2>3}

${empty requestScope["helloworld"]} :判断是否为null或者不存在 如果是则true

EL的隐式对象 隐式对象的大小关系

a作用域对象 pageScope< requestScope <sessionScope<applicationScope

如果不指定域对象，则默认会根据从小到大的顺序，依次寻找值 如 ${student}

b参数访问对象：获取表单数据（request.getparamete）《=》${param.参数名}

(request.getparametevalues) <=>${paramValues.参数名}

` c jsp隐式对象 pageContext ： 获取隐式对象

${pageContext.getrequest}:获取request作用域对象

${pageContext.getsession}：获取session作用域对象

jstl

作用：主要用于简化作用域（request，session，application，pagecontext）的传值和取值

jstl.jar standard.jar

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

c:前缀

核心标签库，通用标签库，条件标签库，迭代标签库

a 通用标签库：

c:set/赋值，在某个作用域中个某个变量赋值

<c:set var="" value="" scope="作用域"/>

<c:set target="${requestScope.student}" propertory="sname" value="zxs"/>

可以给不存在的变量赋值，不可以给不存在的对象赋值

<c:set var="y" value="x" scope="request">

${requestScope.y}

<c:out value="${requestScope.m}" default="xkjsk"/>:显示不存在的值，不存在时值为default

<c:out value=""/>

a href="https://www.baidu.com" 百度

<c:out value=" 百度"escapeXml="flase"/>

<c:out value=" 百度"escapeXml="true"/>

escapeXml=“flase”可以实现和超链接一样的效果

<c:remove var="${requestScope.y}" scope="request"/>

<c:if test="${10>2}"/>

过滤器（拦截器）和监听器

作用：用于过滤请求和响应

实现方法：

1.在类上添加注解@WebFilter() 或者 配置web.xml文件（具体配置请查看官网）

2.实现Filter接口

3.重写接口的方法

4.将过滤的操作代码写在doFilter()方法中

web.xml中的配置

c com.baidu.filter.myfilter c/*REQUESTFORWARD

过滤器类的dofitter方法

@Overridepublic void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)throws IOException, ServletException {// TODO Auto-generated method stubSystem.out.println("过滤请求");//放行chain.doFilter(request, response);System.out.println("过滤响应");}

过滤器链

filter2

1221

监听器

1监听request，session，application对象

public class mylisten implements ServletContextListener,ServletRequestListener,HttpSessionListener{

2监听request，session，application对象的属性（增删改）

public class mylisten2 implements
HttpSessionAttributeListener,ServletRequestAttributeListener,ServletContextAttributeListener

web配置

com.baidu.listen.mylisten2

****

实例

使用session监听器显示当前在线人数

原理：通过监听判断，每次生成一个session对象，则增加一个application加一当session对象

销毁时（session对象30分钟无任何操作）application-1

session的钝化和活化

session钝化：将session对象保存在硬盘相当于序列化

session活化：将session对象从硬盘中获取相对于反序列化

场景1:服务器内存容不下session对象时，可以将session钝化

场景2:服务器需要重启时，session保存到硬盘里，重启时就可以从硬盘读取session对象

jndi和tomcat连接池

jndi可以保证在配置完后每个项目都能通过name访问

修改tomcat/conf/context.xm加下面代码

```
<% Context ctx=new InitialContext();
String testjndi=(String)ctx.lookup("java:comp/env/jndiName");
out.print("
");
out.print(testjndi);
%>
```

数据库连接池

tomcatdbcp, dbcp c3p0 ,druid

可以用数据源（javax.sql.DataSource）管理连接池

tomcat连接池：

先在context.xml配置

```
<Resource name="user2" auth="Container" type="javax.sql.DataSource"

        maxActive="100" maxIdle="30" maxWait="10000"

        username="root" password="123456" driverClassName="com.mysql.jdbc.Driver"

        url="jdbc:mysql://localhost:3306/user"/>
```

配置完context.xml后配置web.xml连接池数据源

```
<resource-ref>    <res-ref-name>user2</res-ref-name>    <res-
type>javax.sql.DataSource</res-type>    <res-auth>Container</res-auth>
</resource-ref>
```

tomcatdbcp:

servlet中的代码： `Context initCtx = new InitialContext();//初始化查找命名空间Context ctx = new InitialContext();Context envContext = (Context)ctx.lookup("java:/comp/env");//参数jdbc/mysqlds为数据源和JNDI绑定的名字 DataSource ds = (DataSource)envContext.lookup("user2");`

dbcp连接池

步骤：

1导入jar common-dbcp和common-pool

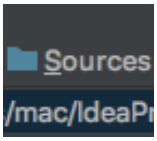
配置啥的看百度

总结intelite idea踩过的坑

1普通文件夹里面的类无法访问library中的类

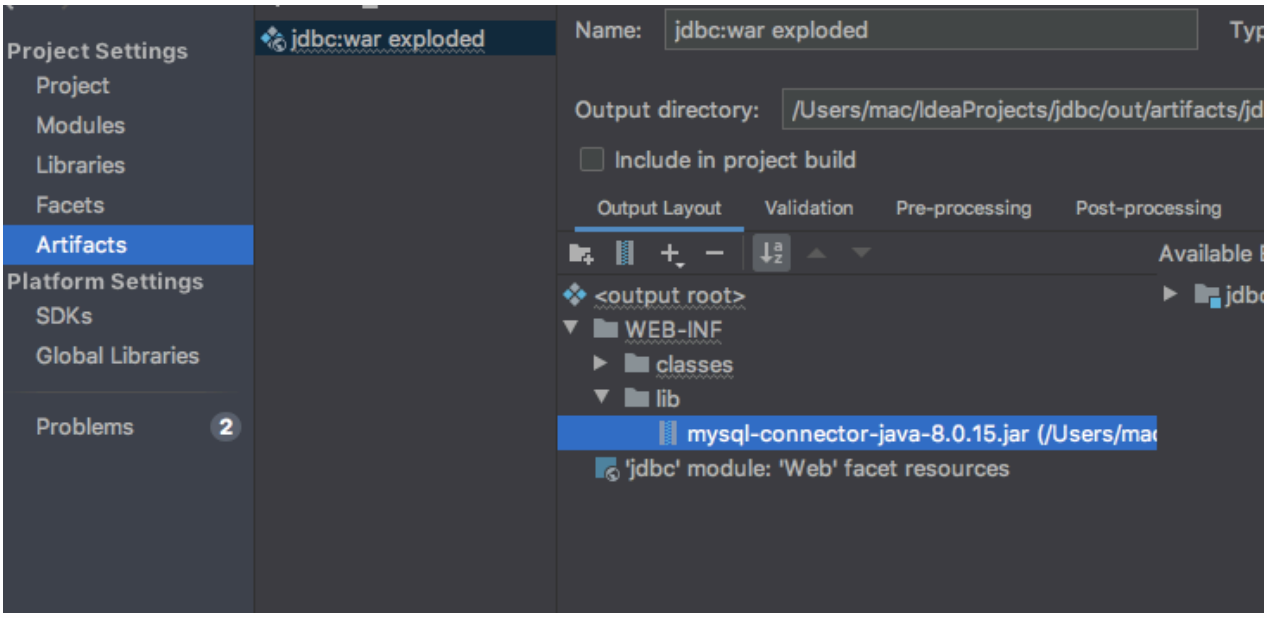
解决办法，直接将类写到src或者直接给文件夹 右键

 然后给这个文件夹，： 点击sources



2:在web项目中：导入jar

在web-inf下新建lib和classes并且在



地方处新建lib文件夹并选择该文件夹点击中间的加号添加该jar

添加完之后jar这样该jar就会在运行期间有效

如果需要jar在整个程序运行的过程中有效，则需要将jar复制到src下，然后右键add as library

这样jar就会在整个程序运行的过程中有效

Dbutil用法

引入jar包

commons-dbutils-1.7.jar

方法1:

1使用自己实现的ResutlsetHander myhandler,

```
import org.apache.commons.dbutils.ResultSetHandler;import
java.sql.ResultSet;import java.sql.SQLException;import
java.util.ArrayList;import java.util.List;public class myhander implements
ResultSetHandler {    @Override    public Object handle(ResultSet resultSet)
throws SQLException {        List<student>list=new ArrayList<student>();
   while (resultSet.next()){            int id=resultSet.getInt(1);
 String name=resultSet.getString(2);            String
sex=resultSet.getString(3);            student s=new student(id,name,sex);
         list.add(s);        }        return list;    }}
```

```
 public void testfindall() throws SQLException {        Connection
connection=Dbcputil.getDataSource().getConnection();        QueryRunner
queryRunner=new QueryRunner(Dbcputil.getDataSource());//
Map<String,Object> queryRunner.query(connection,"select * from student where
id=1",new MapHandler(),(Object[])null);//        for (){////        }
myhander myhander=new myhander();``    //通过重写ResutlsetHander得到返回我们需要
的类型值      List<student>list= (List<student>) queryRunner.query("select *
from student",myhander);    }
```

**

**

方法2:

2使用**ArrayHandler返回第一行数据***

将 queryRunner.query方法中的Resultsethandler改成***ArrayHandler即可，不演示***

①**ArrayHandler：　将查询结果的第一行数据，保存到Object数组中*** ②****ArrayListHandler**　将查询的结果，每一行先封装到**Object**数组中，然后将数据存入**List**集合**** ③**BeanHandler　将查询结果的第一行数据，封装到user对象** ④**BeanListHandler　将查询结果的每一行封装到user对象，然后再存入List集合** ⑤**ColumnListHandler　将查询结果的指定列的数据封装到List集合中** ⑥**MapHandler　将查询结果的第一行数据封装到map结合（key列名，value列值）** ⑦**MapListHandler　将查询结果的每一行封装到map集合（key列名，value列值），再将map集合存入List集合** ⑧**BeanMapHandler　将查询结果的每一行数据，封装到User对象，再存入mao集合中（key列名，value列值）**

建议用自己实现的类

用queryrunner 实现增删改

```
public  void testzengjia() throws SQLException {    Connection
connection=Dbcputil.getDataSource().getConnection();    QueryRunner
queryRunner=new QueryRunner(Dbcputil.getDataSource());    String sql="insert
into student values(?,?,?)";    //增加一条数据     int update =
queryRunner.update(sql,1, "狗蛋","男");}
```

增删改都是用queryRunner.update（）方法

```
public void testupdate() throws SQLException {//初始化queryRunner  参数为数据源
QueryRunner queryRunner=new QueryRunner(Dbcputil.getDataSource());    String
sql="update student set sname=? where id=?";    //第一个参数是sql语句，第二个是params
queryRunner.update(sql,"里斯",1);}
```
总结：queryrunner.query()返回的是查询结果，跟参数ResultserHandler 有关，参数都列举在上面了

queryrunner.update（）为怎删改，

当queryrunner中五connection时，不需要关闭connection之类的，内部已经写好了。