1. 数据库

登录：mysql -uroot -p

案例1-单表的crud---增删改查

关系型数据库：二维表：实体-实体之间关系

非关系型数据库：存放的是对象（redis）No-sql

sql：管理数据库

sql的分类：

DDL:

数据定义语言---增删改

操作对象:数据库和表

关键词：create alter drop

操作数据库：

创建数据库：create database 数据库名称

删除数据库：drop database 数据库名称

show databases;

操作表：

创建表：

create table 表名(字段描述,字段描述)；

字段描述：字段名称 字段类型[约束]

例如：

create table user(

id int primary key auto\_increment,

username varchar(20)

);

修改表：

格式：

alter table 表名...

修改表名：

alter table 旧表名 rename to 新表名；

修改字段：

alter table 表名 add [column] 字段描述；

修改字段名：

alter table 表名 change password pwd vachar(20);

修改字段描述：

alter table 表名 modify 字段名称 字段类型 [约束];

删除字段：

alter table 表名 drop 字段名；

删除表：

drop table 表名；

常用命令：

切换数据库：use 数据库名称

查看所有表：show tables;

查看表结构：desc 表名

查看建表语句：show create table 表名；

DML:

数据操作语言---增删改

操作对象：记录(行)

关键词：

insert update delete

插入：

格式1：

insert into 表名 values(字段值1，字段值2，...,字段值n);

注意：

默认插入全部字段，

必须保证values 后面的内容的类型和表结构中的一致

若字段类型为数字，可以省去引号

例如：

insert into user values(1,'tom')

insert into user values('2','tom')

insert into user values('3')-----错误的

格式2：

insert into 表名(字段名1，字段名2..) values (字段值1，...)

注意：

插入指定字段，

保证value后面的内容和顺序和表名后面的字段的类型和顺序保持一致

例如：

insert into user (username,id) values ('jack',4);

insert into user (username) values ('jack',5); ---error

修改：

格式：

update 表名 set 字段名=字段值，字段名2=字段值2..[where 条件];

例如：

update user set username='jerry' where username='jack';---或者id=...

删除：

格式：

delete from 表名 [where 条件]

删除有物理删除和逻辑删除（常用），

逻辑删除一般会在表中添加一个字段，（isDel 若值为1 ，代表删除了。若为0，代表没有删除，此时删除操作变成了更新操作）

例如：

delete from user where id = '2';

DQL:

数据查询语言（非官方）

操作记录

关键词：select

格式：

select ... from 表名 where 条件 group by 分组字段 having 条件 order by 排序字段 asc|desc

执行顺序：

1.确定数据来自哪张表 from

2.是否主要筛选 where

3.是否需要分组 group by

4.分组后是否需要筛选 having

5.是否需要排序 order by

6.确定显示哪些数据 select

假设表：

创建商品表：

create table products(

pid int primary key auto\_increment,

pname varchar(20),

price double,

pnum int,

cno int,

pdate timestamp

);

insert into products values (null,'泰国大榴莲',98,12,1,null);

查询：

1.查询所有的商品

select \* from products;

2.查询商品名和商品价格.

select pname,price from products;

3.查询所有商品都有那些价格. --distinct去重

select distinct price from products;

4.将所有商品的价格+10元进行显示.(别名)

-----给列起别名 格式： 字段名 [as] 别名

select price+10 from products;

select price+10 新价格 from products;

select price+10 '新价格' from products; --有空格时必须加‘’

条件查询：

1.查询商品名称为香蕉的商品所有信息：

SELECT \* FROM `products` where pname='香蕉'

2.查询商品价格>60元的所有的商品信息:

SELECT \* from `products` WHERE price>60

3.查询商品名称中包含”新”的商品

------模糊匹配

------格式：字段名 like "匹配规则"

匹配内容：

"啊" 值为啊

"%啊" 值以啊结尾

"啊%" 值以啊开头

"%啊%" 包含啊

匹配个数：

"\_" 几个下划线代表几个位置

4.查询价格为38,68,98的商品

SELECT \* FROM `products` where price in (38,68,98);

where 后的条件写法：

\* > ,<,=,>=,<=,<>-----不等于

\* like 使用占位符 \_ 和 % \_代表一个字符 %代表任意个字符.

\* in在某个范围中获得值.

\* between 较小值 and 较大值

排序查询:

1.查询所有的商品，按价格进行排序.(asc-升序,desc-降序)

select \* from products order by price desc;

2.查询名称有新的商品的信息并且按价格降序排序.

select \* from `products` where pname like "%新%" order by price desc;

聚合函数:

----对列进行计算，返回值是一个,忽略null值 上面是对行进行计算

\* sum(),avg(),max(),min(),count();

1.获得所有商品的价格的总和：

select sum(price) from products;

2.获得商品表中价格的平均数：

select avg(price) from products;

---round(值，保留小数位数)

select round(avg(price),2) from products;

3.获得商品表中有多少条记录：

select count(\*) from products;

分组：使用group by

单独使用group by时，只显示第一条数据，行数为分组的个数，一般不单独使用

1.根据cno字段分组，分组后统计商品的个数.

select cno,count(\*) from products group by cno;

2.根据cno分组，分组统计每组商品的总数量，并且总数量> 200;

select cno,sum(pnum) from products group by cno;

select cno,sum(pnum) from products group by cno having sum(pnum)>200;

注意：

where 和having区别：

1.where是对分组前的数据进行过滤，having 是对分组后的数据进行过滤

3.where 后面不使用聚合函数，having可以

/////////////////////////////

数据类型：

Java MySQL

byte tinyint

short smallint

int int(!)

long bigint

char/String varchar(!)|char

varchar:可变长度 mysql方言 varchar(20) 存放abc 占三个

char: 固定长度 char(20) 存放abc 占20个

boolean tinyint|int

float|double float|double

double（5,2）该小数长度为5个，小数占2个

java.sql.Date date日期

java.sql.Time time时间

java.sql.TimeStamp timestamp(!) 时间戳 若给定null 数据库存放当前时间

datetime(!) 日期+时间（数据库独有 java没有）

java.sql.Clob(长文本) text(mysql 方言)

java.sql.Blob(二进制) blob

约束：

为了保证数据的有效性和完整性

mysql中常用约束：

主键约束（primary key）,唯一约束（unique）,非空约束（not null），外键约束（foreign key）

primary key：唯一 非空

------一张表只能有一个主键，这个主键可以包含多个字段

1.建表的同时添加约束 格式：字段名称 字段类型 primary key

最常用，只能给一个字段添加主键

2.建表的同时在约束区添加约束 格式： 所有的字段声明完成之后 primary key(字段1，字段2)

2 3 可以给多个字段

create table pk01(

id int,

username varchar(20),

primary key(id，username)

);

3.建表之后，修改表结构添加约束

create table pk02(

id int,

username varchar(20),

);

alter table pk02 add primary key（字段1，字段2...）

多个字段是主键被叫做联合主键，只有都一样的时候，才看做一条记录

唯一约束（了解）：

---------唯一 对null不起作用

1 2 3与上面相同

create table un(

id int unique,

username varchar(20) unique, /////和上面的不同，可以给多个字段添加

);

非空约束（了解）：

create table nn(

id int not null,

username varchar(20) not null

);

///////////////////////////////////////

truncate 清空表

格式：

truncate 表名; 干掉表，重新创建一张空表

和delete from 区别：

delete 属于DML语句，truncate属于DDL语句

delete 逐条删除 truncate干掉表，重新创建一张空表

auto\_increment:

要求：

1.被修饰的字段支持自增 一般为int

2.被修饰的字段必须是一个key,一般是primary key

**DCL**:

数据控制语言

操作对象：用户 事务 权限

2. 案例2-创建多表 ，可以描述表与表之间的关系

需求：

网上商城

实体：用户 商品 分类

常见关系：

一对多：用户-订单，分类-商品

多对多：订单-商品，学生-课程

一对一：

ER图描述实体与实体之间的关系 --实体 属性 关系

一对多：

一方称之为主表或一表，多方称之为从表，为了表示一对多的关系，一般会在多表的一方添加一个字段，字段名称建议为（主表的名称\_id）,字段类型一般和主键的类型保持一致，称这个字段为外键

垃圾数据：

用户删了，订单里还有；

为了保证数据的有效性和完整性，在多表的一方，添加外键约束（不是必须的，也可以通过java程序控制），外键可以是null

格式：

alter table 多表名称 add foreign key （外键名称）reference 主表名称（主键）;

添加了外键约束之后，有如下特点：

1.主表中不能删除从表中已引用的数据

2.从表中不能添加主表中不存在的数据

多对多：

在开发中引入一张中间表，在中间表中存放两张表的主键，一般还会将这两个主键设置成中间表的联合主键（也可以不），，将多对多拆分成两个一对多

为了保证数据有效性和完整性，在中间表上添加两个外键约束即可（不是必须的，也可以通过java程序控制）

一对一：

身份证-人

（1）将两个实体合二为一

（2）在一个表上将这个表的主键设置成外键且添加外键约束

3. 案例3-多表查询

内连接

外链接（左[外]连接 右[外]连接）

子查询

笛卡尔积：多张表无条件的联合查询。没意义

练习:（1）（2）

查询用户的订单,没有订单的用户不显示

SELECT user.\*,orders.\*

FROM `user`,orders

WHERE user.id=orders.user\_id

SELECT user.username,orders.\*

FROM `user`inner join orders

ON user.id=orders.user\_id

查询所有用户的订单详情

左外连接，所有用户---用户在左

SELECT user.\*,orders.\*

FROM user LEFT JOIN orders

ON user.id=orders.user\_id

查询所有订单的用户详情

右外连接，--所有订单，订单在右

SELECT orders.\*, user.\*

FROM `user` RIGHT JOIN orders

ON user.id=orders.user\_id

（1）内连接：

格式1：显式的内连接

select a.\*,b.\* from a [inner] join b on ab连接的条件

格式2：隐式的内连接

select a.\*,b.\* from a,b where ab连接的条件

（2）外连接（重要）

左外连接：

select a.\*,b.\* from a left [outer] join b on 连接条件;

先展示join左边的（a）表的所有数据，根据条件关联查询join 右边的表（b）,符合条件则展示出来，不符合以null值展示。

右外连接：

select a.\*,b.\* from b right [outer] join a on 连接条件;

先展示join右边的（a）表的所有数据，根据条件关联查询join 左边的表（b）,符合条件则展示出来，不符合以null值展示。

（3）子查询（重要）

给表起别名：表 [as] 别名

练习:（3）

查看用户为李四的订单详情

SELECT orders.\*

from orders

WHERE user\_id =(

SELECT id

from `user`

WHERE username = '李四'

)

查询出订单的价格大于300的所有用户信息。

SELECT user.\*

FROM `user`

WHERE id in(

SELECT user\_id

FROM orders

WHERE totalPrice>300

)

查询订单价格大于20的订单信息及相关用户的信息

内连接：

SELECT orders.\*,user.\*

FROM orders ,`user`

WHERE orders.user\_id=user.id AND orders.totalPrice>20

子查询：

SELECT user.\*,tmp.\*

FROM user,

(SELECT orders.\*

FROM orders

WHERE totalPrice>20 )AS tmp

WHERE `user`.id=tmp.user\_id

4. JDBC

java操作数据库，一套接口，

驱动----->（jdbc的一套实现类 由数据库厂商提供)

jdbc作用：

连接数据库

发送sql语句

处理结果

jdbc操作步骤：

1.导入驱动jar 包（驱动）

2.注册驱动

Class.forname("com.mysql.jdbc.driver")

3. 获取连接

有时候出现中文字符问题：

Connection conn =DriverManager.getConnection("jdbc:mysql://localhost:3306/day07?useUnicode=true&characterEncoding=utf8", "root", "123456");

或者已经将my.ini修改 修改好了

4.编写sql

5.创建语句执行者

PreparedStatement st = conn.prepareStatement(sql)

6.设置参数

st.setXXX()

7.执行sql

ResultSet rs= st.execuQuery();

int i = st.execuUpadate()

8.处理结果

if(rs.next()) re.getXXX(int | string)

9.释放资源

案例1-通过jdbc 完成单表的crud操作

IDE设置；

使用junit单元测试

1.方法是 public void xxx{}

2.在方法上添加 @Test

3. ctr+1 快速锁定错误

4.在方法上右键 run as --》junit就可以执行方法 （底层是main）

5. JDBC-api:

所有的包 都是java.sql 或者javax.sql

DriverManager:管理了一组jdbc的操作类

常用方法：

了解：注册驱动

static void registerDriver(Driver driver):

通过查看com.mysql.jdbc.Driver的源码， 有如下：

static {

try {

java.sql.DriverManager.registerDriver(new Driver());

} catch (SQLException E) {

throw new RuntimeException("Can't register driver!");

}

驱动注册了两次，我们只需将静态代块执行一次，类被加载到内存会执行静态代码，并且只执行一次，现在只需将类加载到内存即可：

方式1：掌握

Class.forName("全限定名") //包名+类名 com.mysql.jdbc.Driver

方式2：

类名.class;

方式3：

对象.getClass() ;

掌握：获取连接

static Connection getConnection(String url, String user, String password)

参数1：

协议：数据库类型:子协议：参数

jdbc:mysql://localhost:3306/数据库名称

参数2 3：账号密码

Connection 连接 接口

常用方法：

获取语句执行者：

\*Statement createStatement() 普通的语句执行者 会出现sql注入

PreparedStatement prepareStatement(String sql) ：获取预编译语句执行者

\*CallableStatement prepareCall(String sql)：获取调用存储过程的语句执行者

了解：

setAutoCommit（false） 手动开启事务

commit():提交事务

rollback():事务回滚

Statement:语句执行者 接口

PreparedStatement:预编译语句执行者 接口

常用方法：

(1)设置参数：

setXXX(int 第几个问号,Object 实际参数)；

常见的方法:

setInt

setString

setObject

(2)执行sql:

ResultSet executeQuery():执行r语句 返回值：结果集

int execuUpdate():执行cud语句 返回值：影响的行数

ResultSet ：结果集 接口

执行查询语句之后返回的结果

常用方法：

boolean next():判断是否有下一条记录，若有返回true 且将光标移到下一行，若没有，则返回false

获取具体内容：

getXXX(Int | String)

若参数为Int ：第几列

若参数为String： 列名

例如：

获取cname的内容可以通过

getString（2）

getString("cname")

常用方法：

getInt

getString 也可以获取int值

getObject 可以获取任意值

///////////////////////////////////////////////////

\*\*\*快捷键总结：

ctrl+shift+f :格式化代码

ctrl+o：整理包 这个类内的所有方法都会显示

alt+/ ：提示

ctr+1：快速锁定错误

alt+shift+a: 块编辑 不要用中文输入法

ctrl+2 L

ctrl+shift+X :变大写

ctrl+shift+y:变小写

alt+ctrl+向下： 向下复制一行

shift+enter :向下添加一个空行

ctrl+d：删除一行

alt+向下:向下移动一行

ctrl+o 调出他的所有的方法

////////////////////////////////////////////////////////

常见的配置文件格式：

1.properties

key=value; //里面都是字符串 不用再加引号

回车 之前不能加空格

2.xml

若 配置文件为properties,并且放在src目录,可以通过ResourceBundle工具快速获取里面的配置信息

步骤：

1.获取ResourceBundle 对象：

static ResourceBundle getBundle("文件名称不带后缀名")

2.通过ResourceBundle 对象获取配置信息：

String getString(String key)：通过执行key获取指定value

6. 案例2-通过连接池（数据源）优化操作

使用jdbc的时候 每操作一次都需要获取连接（创建） 用完之后把连接释放掉（销毁） 通过连接池来优化crud操作

技术：连接池

管理数据库的连接

作用：

可以提高项目性能，

就是在连接池初始化的时候存入一定数量的连接 用的时候通过方法获取 不用的时候归还连接即可

所有的连接池必须实现一个接口，javax.sql.DataSource接口

获取连接的方法：

Connection getConnection()

归还连接的方法就是 释放资源的方法，

Conn.close();

常用的连接池：

DBCP(了解) ：

apache

步骤:

1.导入jar包（dbcp.jar ,pool.jar）

2.使用API

a.硬编码

// 创建连接池

BasicDataSource ds = new BasicDataSource();

// 配置信息等

ds.setDriverClassName("com.mysql.jdbc.Driver");

ds.setUrl("jdbc:mysql:///day07?useUnicode=true&characterEncoding=utf8");

ds.setUsername("root");

ds.setPassword("123456");

b.配置文件

Properties prop = new Properties();

prop.load(new FileInputStream("src/dbcp.properties"));

DataSource ds =new BasicDataSourceFactory().createDataSource(prop);

没有自动回收的功能

C3P0(❤)：

hibernate spring使用

有自动回收空闲连接的功能

步骤：

1.导包

2.使用API

a.硬编码

new ComboPooledDataSource()

b.配置文件

配置文件的名称：c3p0.properties 或者 c3p0-config.xml

配置文件的路径：src下

编码：

new ComboPooledDataSource() 就可以 //使用默认的配置

new ComboPooledDataSource(String configName)//使用命名的配置，若找不到使用默认的

增强方法：

1.继承 需要父类的源码 局限

2.装饰者模式 （静态代理）

步骤：

1.装饰者和被装饰者实现同一个接口 或继承同一个类

2.装饰者要有被装饰者的引用 (通过构造器 引用写成被装饰者的父类，这样不仅可以装饰这个 被装饰的层级的)

3.对需要增强的方法进行增强

4.对不需要加强法人方法调用原来的方法

3.动态代理

7. static 关键字：

1.static 关键字的用途：

方便在没有创建对象的情况下来进行调用（方法/变量）。

(1)static方法：

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。

(2)static变量

静态变量被所有的对象所共享，在内存中只有一个副本，它当且仅当在类初次加载时会被初始化

(3)static代码块

static关键字还有一个比较关键的作用就是用来形成静态代码块以优化程序性能。static块可以置于类中的任何地方，类中可以有多个static块。在类初次被加载的时候，会按照

static块的顺序来执行每个static块，并且只会执行一次。

8. 案例3-使用 dbutils完成crud

dbutils: apache 的一个工具类

使用步骤：

1.导入jar包（commons-dbutils-1.4.jar）

2.创建一个queryRunner类

queryRunner 作用：操作sql

构造方法 new QueryRunner(DataSource ds);

3编写sql （只能带问号的那种）

4.执行sql

query() r操作

update() cud操作

核心类：

QueryRunner: 操作sql语句

构造器：

new QueryRunner(DataSource ds);

常用方法：

query() r操作

update() cud操作 在update里可以找到？设置参数的方法

Dbutils: 类 释放资源 控制事务

closeQuietly(conn):内部处理了异常

commitAndClose(Connection conn): 提交事务 并释放连接

ResultSetHandler:封装结果集 接口 查询

9个实现类：

ArrayHandler, 将查询结果的第一条记录封装成数组 返回

ArrayListHandler, 将查询结果的每一条记录封装成数组，将每一个数组放入list返回

❤❤BeanHandler：将查询结果的第一条记录封装成指定的bean对象，返回

❤❤BeanListHandler, 将查询结果的每一条记录封装成指定的bean对象，将每一个bean对象放入list返回

ColumnListHandler,将查询结果的指定一列发入list 返回

KeyedHandler,

MapHandler, 将查询结果的第一条记录封装成map,字段名为key 值为value 返回

❤MapListHandler, 将查询结果的每一条记录封装成map集合, 将每一个map集合放入list返回

❤ScalarHandler：针对于聚合函数 例如 count() 返回long值

：map.getClass().getName() 可以返回map类型

9. xml:

可扩展的标签语言

标签自定义

作用：存储数据 （配置文件）

书写规范：

1.区分大小写

2.必须有一个根标签

3.标签必须关闭

<xx></xx>

<xx/>

4.属性必须用引号引起来

<xx att="value"/>

5.标签体中的空格或者换行或者制表符等内容都是作为数据存在的

6.特殊字符必须转义 < > &

满足以上规范的文件称之为是一个格式良好的xml文件 可以通过浏览器浏览

后缀名 .xml

组成：

声明：第一行 顶格写

元素（标签）：

<xx></xx>

<xx/>

要求：

1.必须关闭

2.标签名中不能 xml Xml XML 等开头

3.标签名不能出" " 和:等特殊字符

属性：

格式：

<xx 属性名="属性值"></xx>

属性必须用引号引起来

注释:<!-- -->

CDATA:

xml文件特殊字符必须转义

1.---------三部分 例如<a> 写为 &lt;a&gt;

2.<![CDATA[ 可以原样输出的。。。。。]]>

xml解析：

解析方式：

1.sax 逐行解析 只能查询

2.dom 一次性将文档加载到内存 形成dom树 可以对dom树进行crud

解析技术：

 常见的解析开发包：

 JAXP：sun公司提供支持DOM和SAX开发包

 JDom：dom4j兄弟

 jsoup：一种处理HTML特定解析开发包

❤dom4j：比较常用的解析开发包，hibernate底层采用。

dom4j技术进行查询操作：

使用步骤：

1.导入jar包

2.创建一个核心对象 SAXReader

new SAXReader

3.将xml文档加载到内存中形成一颗树

Document doc=reader.read(文件名)

4.获取根节点

Element root = doc.getRootElement();

5.通过根节点就可以获取其他节点（文本节点 属性节点 元素节点）

获取所有子元素

List<Element> list = root.elements()

获取元素的指定属性内容

String value = root.attributeValue("属性名")

获取子标签标签体 ：遍历list 获取到每一个子元素

String text = ele.elementText("子标签名称")

缺点是 若想找到一个嵌套深的 则需要一步步获取

xpath 解析技术：

依赖于dom4j

使用步骤：

1.导入jar包(dom4j 和 jaxen-1.1-beta-6.jar)

2.加载文件到内存

3.使用API

selectNode("表达式")

selectSingleNode("表达式")

4.eleSing.getText() 获取中间文本

表达式的写法：

/ 从根节点选取。

// 从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置

获取属性的值：

例如 //AAA[@属性名="属性值"]

10. 案例1-编写配置文件 编写一个服务器软件 按照指定的全限定名 根据路径 让服务器创建这个对象，调用指定的方法

解析xml

通过全限定名创建一个对象 调用方法 ----------反射

11. 反射

1.获取对应的class对象

方式1：❤

Class clazz = Class.forname("全限定名")

方式2：

Class clazz = 类名.Class();

方式3：

Class clazz = 对象.getClass();

{

Class类也是类的一种，与class关键字是不一样的。

手动编写的类被编译后会产生一个Class对象，其表示的是创建的类的类型信息，而且这个Class对象保存在同名.class的文件中(字节码文件)，比如创建一个Shapes类，编译Shapes类后就会创建其包含Shapes类相关类型信息的Class对象，并保存在Shapes.class字节码文件中。

每个通过关键字class标识的类，在内存中有且只有一个与之对应的Class对象来描述其类型信息，无论创建多少个实例对象，其依据的都是用一个Class对象。

Class类只存私有构造函数，因此对应Class对象只能有JVM创建和加载

Class类的对象作用是运行时提供或获得某个对象的类型信息，这点对于反射技术很重要。

}

2.通过Class对象创建一个实例对象(相当于 new 类())

Object clazz.newInstance();

3.通过Class 对象 获取一个方法

Method method = clazz.getMethod(“方法名”，Class .... param)

param为参数类型

4.让方法执行

method.invoke(a,10,20)

method.invoke(Object 实例对象，Object ...参数) ==a

Object ...参数 ：该方法运行时需要的参数 ==10,20

12. xml约束：

作用：规定xml中可以出现哪些元素及哪些属性 ，以及出现的顺序。

约束的分类：

DTD约束： structs hibernate

需要导入dtd 文件

和xml的关联：

方式1：内部关联

格式：

<!DOCTYPE 根元素名 [dtd语法]>

\*dtd语法：

元素：

<Element 元素名称 数据类型 | 包含的内容>

数据类型：

#PCDATA 普通文本 使用的时候用（）引起来

包含内容：

该元素下可以出现哪些元素 用（）引起来

符号：

\* 任意次

？ 0或1次

+ 至少1次

| 或者

() 分组

， 逗号表示顺序

属性：

格式：

<!ATTLIST 元素名 属性名 属性类型 属性是否必须出现>

属性类型：

ID：唯一

CDATA：普通文本

属性是否必须出现：

REQUIRED：必须出现

IMPLIED:可以不出现

一个xml文档只能出现一个dtd约束

方式2：外部关联 系统关联

格式：

<!DOCTYPE 根元素名 SYSTEM "约束文件位置">

方式3：外部关联 公共关联

格式：

<!DOCTYPE 根元素名 PUBLIC "约束文件的名称" "约束文件位置">

SCHEMA约束： tomcat 项目 spring

一个xml文档能出现多个SCHEMA约束

和xml的关联：

格式：

<根标签 xmlns="..." ...>

<根标签 xmlns：别名="..." ...>

名称空间：

关联约束文件

规定元素是来源于哪个文件

例如：

一个约束文件中规定table （表格） 表格有属性 row col

另一个约束文件table （桌子） 桌子有属性 width height

同一个xml引入了两个table 属性？为避免这种情况 可以给其中一个约束起个别名 使用时未加别名代表来源于没有别名的约束文件

例：table (表格) xmlns:a="..."

a:table

xml 只有一个不起别名

注意：

SCHEMA约束本身也是一个xml文件

关于xml掌握：

根据约束写出name value 属性即可 ，知道标签出现的位置

Alt+/ 或者 F2 提示写出内容即可

13. 案例2-通过eclispe发布自己的项目

eclispe

tomcat :服务器

web 项目

////////////////////////////////////////////////

服务器分类：硬件服务器 和 软件服务器

web 服务器：提供资源供别人访问

web ： 网页 资源

web资源分类：

静态web: html css ...

动态web: servlet jsp php .net

web 通信机制：请求响应机制

tomcat :支持servlet 和 jsp规范的服务器

配置：

1.启动一闪而过 检查JAVA\_HOME

2.端口冲突

修改tomcat端口号 conf server.xml 修改port后面的值就可以 1024以下留给系统用 80除外 留给HTTP用的

3.CATALINA\_HOME可能出现 启动多个版本只能启动一个 原因配置好了。。删除

目录：

❤webapps: 存放项目的目录

❤work: 存放jsp文件在运行时产生的java 和 class文件

web 项目的目录结构：❤❤❤

myweb（项目名称） web 2.5版本标准的目录结构

|

|----------html css js image 等目录或者文件

|

|----------WEB-INF(特点：通过浏览器直接访问不到 目录，但是通过java程序可以)

| |

| |-------lib(项目第三方jar包)

| |-------classes(存放的是我们自定义的java 文件生成的.class文件)

| |-------web.xml(当前项目的核心配置文件)

| |

|

访问路径：

http://主机：端口号/项目名称/资源路径

项目发布方式：（虚拟目录映射）

❤1.项目放到tomcat/webapps下

\*\*2.server.xml修改

host标签下，添加如下代码

<Context path = "/项目名称" docBase = "项目目录"/>

\*\*3. tomcat/conf/引擎目录/主机目录下 新建一个xml文件

文件名就是项目名 文件内容：<Context docBase ="G:..."/>

//////////////////////////////////////////////////

eclispe和tomcat整合 ❤❤❤

day08doc参考

修改eclispe的servers

通过eclispe 发布项目：

1.创建一个项目 （动态的web项目）

2.web版本2.5

3.java 文件在src

网页图片在webContent下

4.发布项目

14. servlet:

htttp协议：要求掌握一些头信息

超文本传输协议 ：规定数据的格式

浏览器---服务器发送：请求

请求：

请求行（第一行）

格式：请求方式 访问的资源 协议/版本

例：GET day08/1.html Http/1.1

请求方式:get post

get 有参数 post无

get参数大小有限制

get 没有请求体 post有 参数在请求体中

请求头（第二行到空行之前）

格式：key:value(value 可以有多个值)

常见的请求头：

请求头

Accept: text/html,image/\* --支持数据类型

Accept-Charset: ISO-8859-1 --字符集

Accept-Encoding: gzip --支持压缩

Accept-Language:zh-cn --语言环境

Host: www.itcast.cn:80 --访问主机

If-Modified-Since: Tue, 11 Jul 2000 18:23:51 GMT --缓存文件的最后修改时间

Referer: http://www.itcast.com/index.jsp --来自哪个页面、防盗链

User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)

Cookie

Connection: close/Keep-Alive --链接状态

Date: Tue, 11 Jul 2000 18:23:51 GMT --时间

掌握：

Referer

User-Agent 浏览器内核

Cookie

If-Modified-Since

请求体（空行以下）：

服务器---浏览器返回：响应

响应行：协议/版本 状态码

状态码说明 ：

200 成功

302 重定向

304 读缓存

404 用户操作错误

500 服务器内部异常

响应头

格式：key:value(value 可以有多个值)

常见的头：

Location: http://www.it315.org/index.jsp --跳转方向 302一起使用的

Server:apache tomcat --服务器型号

Content-Encoding: gzip --数据压缩

Content-Length: 80 --数据长度

Content-Language: zh-cn --语言环境

Content-Type: text/html; charset=GB2312 --数据类型

Last-Modified: Tue, 11 Jul 2000 18:23:51 GMT --最后修改时间

Refresh: 1;url=http://www.it315.org --定时刷新

Content-Disposition: attachment; filename=aaa.zip --下载

Set-Cookie:SS=Q0=5Lb\_nQ; path=/search

Expires: -1 --缓存

Cache-Control: no-cache --缓存

Pragma: no-cache --缓存

Connection: close/Keep-Alive --连接

Date: Tue, 11 Jul 2000 18:23:51 GMT

掌握：

Content-Type Location Last-Modified Refresh Content-Disposition Set-Cookie

响应体：

页面展示的信息

15. 案例1-完成登录案例

表单 servlet request reponse

表单:

收集用户数据 必须有name属性 提交地址action post

servlet：

动态的 web开发技术 本质就是一个类 运行在服务器端的一个java小程序 处理业务逻辑 生成动态的web内容

编写servlet 步骤：

1.编写一个类

a.继承HttpServlet

b.重写 doGet 或者 doPost 方法

2.编写配置文件（web-inf/web-xml）

a.注册servlet

b.绑定路径

3.访问

http://主机：端口号/项目名称/路径

------------tomcat 两个jar包重要：servlet-api jsp-api

接受参数：

格式：key=value

String value = request.getParameter("key")

往浏览器回写内容：

reponse

reponse.getWriter().print("success")

处理相应数据的乱码：

resp.setContentType("text/html;charset=utf8");

实现：

1.数据库 和表

2.工程

3.页面:

修改login.html

给用户名 密码添加name 属性

修改表单的action 属性:

action = "http://localhost/day09/login"

添加method属性：

method = "post"

4.导入jar包

驱动connection.jar dbutils(工具类) c3p0(连接池)

5.导入工具类和配置文件

DataSourceUtils c3p0-config.xml(注意修改数据库名称)

6.创建servlet (LoginServlet : 路径 /login)

接受用户名和密码

调用service层（UserService）完成登录操作

提示信息

7.UserService

login（username，pwd）

调用dao

8.dao:

通过用户名和密码查询数据库

login.html(用户名 密码 提交)

浏览器 <---->

loginServlet（接受username pwd ,调用userservice.login(username pwd) 返回值User user 返回值是一个bean 判断user是否为空）

<------>

UserService（login（name，pwd）{dao.getUserBynameAndPwd（name,pwd）}）

<------>

UserDao(getUserBynameAndPwd {查询数据库})

<------>

DB

16. servlet总结:

servlet 体系结构：

Servlet：接口

|

GenericService:抽象类

|

HttpServlet 抽象类

|

自定义Servlet

servlet常用方法：

void init (ServletConfig config):初始化

void service (ServletRequest request,ServletResponse response):服务 处理业务逻辑

void destory():销毁

ServiceConfig getServiceConfig():获取当前servlet 的配置对象

GenericService常用方法：

除了service方法没有显示 其他都已实现

空参的Init() 若我们想对servlet 进行初始化操作 重写这个init()方法即可

HttpServlet常用方法：

service 做了实现，把参数强转，调用了重载的service 方法

重载的service方法 获取请求的方式，根据请求方式的不同调用相应的 doxxx()方法

doGet 和 doPost方法

servlet生命周期❤❤❤：

void init (ServletConfig config):初始化

\* 初始化方法

\* 执行者：服务器

\* 执行次数：1

\* 执行时机：默认 第一次访问时

void service (ServletRequest request,ServletResponse response):服务 处理业务逻辑

\* 服务方法

\* 执行者：服务器

\* 执行次数：请求一次 执行一次

\* 执行时机：请求来的时候

void destory():销毁

\* 销毁方法

\* 执行者：服务器

\* 执行次数：只销毁一次

\* 执行时机：当servlet被移除 或者服务器正常关闭的时候

servlet是单实例多线程

默认第一次访问的时候 服务器创建servlet,并调用init()实现初始化操作，并调用一次service()方法，每当请求来的时候，服务器创建一个线程，调用service()方法执行自己的业务逻辑

当servlet被移除的时候 服务器正常关闭的时候 服务器调用自己的servlet的destroy方法实现销毁操作。

配置URL-pattern

【完全路径匹配】

\* 以/开头 如:/aaa /aaa/bbb

【目录匹配】

\* 以/开头 以\*结尾 如:/\* /aaa/\* /aaa/bbb/\*

【扩展名匹配】

\* 不能以/开始的 需要以\*开始 如：\*.jsp \*.do \*.action

完全路径匹配 > 目录匹配 > 扩展名匹配

一个路径对应一个servlet 一个servlet对应多个路径

练习:

有如下的一些映射关系：

Servlet1 映射到 /abc/\*

Servlet2 映射到 /\*

Servlet3 映射到 /abc

Servlet4 映射到 \*.do

问题:

当请求URL为“/abc/a.html”，“/abc/\*”和“/\*”都匹配，哪个servlet响应

Servlet引擎将调用Servlet1。

当请求URL为“/abc”时，“/\*”和“/abc”都匹配，哪个servlet响应

Servlet引擎将调用Servlet3。

当请求URL为“/abc/a.do”时，“/abc/\*”和“\*.do”都匹配，哪个servlet响应

Servlet引擎将调用Servlet1。

当请求URL为“/a.do”时，“/\*”和“\*.do”都匹配，哪个servlet响应

Servlet引擎将调用Servlet2.

当请求URL为“/xxx/yyy/a.do”时，“/\*”和“\*.do”都匹配，哪个servlet响应

Servlet引擎将调用Servlet2。

servlet标签中有一个子标签：load-on-startup

作用：用来修改servlet的初始化时机

取值：正整数 值越大 优先级越小

当自己配置的文件里面没有指定配置的话 会查找tomcat conf 的web.xml 默认的servlet处理（404 500 405.。。。。）

路径的写法：

相对路径：

../

绝对路径：

1.带主机+协议（访问站外资源 百度。。不在一个主机上）

http://www.itcost.com/xxx

http://localhost:8080/day09/hello

2.不带主机和协议

/day09/hello（经常使用）

错误的地方：

1.访问地址 http://localhost:8080/day09/life.html 没有写.html

单写/day09/life 指的是（提交到的）对应的servlet地址

17. 案例2-登录失败时提示用户名密码不匹配 3秒跳转登录页面

定时刷新

refresh

常见的响应头-refresh

响应头格式：

refresh：秒数;url=跳转的路径

设置响应头：

response.setHeader(String key;String value);设置字符串形式的响应头

response.addHeader(String key;String value);追加响应头，若之前设置过这个头，则追加，若没有设置过，则设置

设置定时刷新：

response.setHeader("refresh","3;url=/day09/login.htm");

步骤：

修改业务逻辑 servlet

失败打印信息 追加refresh-----response.setHeader()

18. 案例3-统计登录次数

需求：登录成功后，获取之前登陆的总人次，+1，在访问另一个servlet的时候，显示登录成功的总人次.

servletContext:

上下文 （全局管理者）

方法：

setAttribute(String name, Object object) //设置值

getAttribute(String name) //获取值

removeAttribute(String name) //移除值

获取全局管理者：

getServletContext();

步骤：

1.项目启动的时候 初始化登陆次数：

loginServlet的init()方法 中获取全局管理者，值设置为0，放入servContext上

2.登录成功后，在loginServlet中获取全局管理者 获取登录成功的总人次，然后+1，然后将值回存到servletContext中

3. 当访问showServlet的时候，获取全局管理者， 获取登录成功的总人数，然后在页面打印

19. ServletConfig（了解）

这个对象可以获得到Servlet的配置信息.

servletConfig 由服务器创建的，在创建servlet的同时也创建了它，通过servlet的init(ServletConfig config) 将ServletConfig传递给servlet, 由servlet的getServletConfig方法获取

1.获取当前servlet的名称

2.获取当前servlet的初始化参数

❤ 3.获取全局管理者，

方法：

String getInitParameter(String name)

Enumeration getInitParameterNames()

获取所有参数的名称

初始化参数放在web.xml文件 servlet子标签 init-param

❤ServletContext getServletContext()

String getServletName()

20. ServletContext:

上下文 （全局管理者）

一个项目的引用 代表了当前项目。

当项目启动的时候 服务器为每个web项目创建一个servletContext对象

当项目被移除的时候 或者服务器关闭的时候 sevletContext销毁

作用：

1.获取全局的初始化参数 getInitParameter

2.共享资源 （xxxAttribute）

其实就是各个servlet之间的通信

3.获取文件资源

getRealPath

getResourceAsStream

4.其他操作

获取servletContext:

1. getServletContext() //源码其实也是继承实现了ServletConfig

2. getServiceConfig().getServletContext()

常用方法：

1.了解

String getInitParameter(String name) 通过名称获取指定参数值

Enumeration getInitParameterNames() 获取所有的参数名称

在根标签下有一个context-param子标签 存放初始化参数

2.xxxAttribute

3.

getRealPath(String path) :获取文件部署到tomcat上的真实路径（带tomcat路径）

context.getRealPath("/"); E:\Tools\Tomcat\apache-tomcat-7.0.52\wtpwebapps\day09\ 这个目录是部署到tomcat上的路径 WebContent 就是day09/下面的东西

InputStream getResourceAsStream(String path) 以流的形式返回一个文件

4. 获取文件的MIME类型 大类型/小类型

String getMimeType(String 文件名称)

21. 域对象

1. ServletContext

当成一个容器 map集合

常用的方法：

xxxAttribute() ----xxx共三个， get set remove

servlet创建和销毁：

当项目启动的时候 服务器为每个web项目创建一个servletContext对象

当项目被移除的时候 或者服务器关闭的时候 sevletContext销毁

存放：

共享的数据

2. request

创建：一次请求来的时候

销毁：响应生成的时候

作用：

一次请求里面的数据

请求转发：

request.getRequestDispatcher("/dis2(一个内部路径)").forward(request, response);

request生命周期:

一次请求

22. 获取文件路径：

通过类加载器获取文件

---不用依赖servletContext就能获取到的文件目录

---获取的是web-InF的classes (src保存在这里)

PathServlet.class.getClassLoader().getResource("2.txt").getPath();

PathServlet.class.getClassLoader().getResourceAsStream("2.txt")

23. day10

使用 request 和response

24. 案例1- 文件下载案例

技术：

response

文件下载

response:

作用：往浏览器写东西

组成部分：响应行 响应头 响应体

操作响应行：

格式：

协议/版本 状态码 状态码说明

状态码：

1xx:已发送请求

2xx:已完成响应

200：正常响应

3xx:还需浏览器进一步操作

302：重定向 配合响应头:location

304：读缓存

4xx:用户操作错误

404：用户操作错误

405：访问的方法不存在

5xx:服务器错误

500：内部异常

常用方法：

setStatus(int 状态码)：针对于1 2 3

操作响应头：

格式：key/value(value 可以是多个值)

常用的方法：

----设置头

setHeader(String key,String value): 设置字符串形式的响应头

了解：

setIntHeader(String key,int value): 设置整形的响应头

setDateHeader(String key,long value)：设置时间的响应头

----添加头

addHeader(String key,String value) 添加字符串形式的响应头，之前设置过则追加 若没有设置过则设置

了解：

addIntHeader(String key,int value): 添加整形的响应头

addDateHeader(String key,long value)：添加时间的响应头

常用的响应头：

location :重定向

❤方式1：response.sendRedirect("/day10/loc2")

方式2：

response.setStatus(302);

response.setHeader("location", "/day10/loc2");

refresh: 定时刷新

方案1：设置头 refresh

response.setHeader("refresh","3;url=/day10/refresh2.html")

方案2：http的meta 标签

<meta http-equiv="refresh" content="3;/day10/refresh2.html">

content-type: 设置文件的mime类型，设置流的编码及告诉浏览器用什么编码打开

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

content-disposition:文件下载

response.setHeader("content-disposition","attachment;filename="+文件名称)

操作响应体：

页面上要展示的内容

常用方法：

PrintWriter getWriter() 字符流

ServletOutputStream getOutputStream() 字节流

这两个流互斥

自己写的东西用字符流getWriter 其他一概字节流getOutputStream

处理响应中文乱码：

方式1：response.setContentType("text/html;charset=utf-8")

方式2：response.setHeader("content-type","text/html;charset=utf-8")

注意：

两个流互斥

当响应完成之后 服务器会判断一下流是否已经关闭 ，若没有关闭，服务器会关闭（底层使用的缓冲流）

25. 文件下载

下载方式：

1.超链接下载 几乎不使用

<a href="/day10/download/day10.txt">下载day10.txt</a>

若浏览器能解析该资源的mime类型，则打开，若不能 则下载

2.编码下载

通过servlet完成

<a href="/day/10/download?name=day10.txt">下载day10.txt</a>

步骤：

a. 设置文件的mime类型

String mimeType = context.getMimeType(文件名)

response.setContentType(mimeType)

b. 设置下载头信息 content-disposition

response.setHeader("content-disposition","attachment;filename="+filename)

c. 提供流

对口流

response.getOutputStream();

//对拷流 方式1

//获取输入流 输出流

ServletOutputStream os = response.getOutputStream();

InputStream is = context.getResourceAsStream("/download/"+filename);

int len = -1;

byte[] b = new byte[1024];

while((len=is.read(b))!=-1) {

os.write(b, 0, len);

}

os.close(); //关闭的顺序

is.close();

//对拷流 方式2

导入commons-io-1.4.jar， IOUtils.copy(is, os);

26. 通过response+servlet生成验证码

验证码：防止暴力攻击

动态的验证码

27. 案例2-完成注册操作

表单上填写用户数据，点击提交，将所有数据提交到服务器，通过java代码最终保存到数据库中

技术：

表单

request:请求

28. request

作用：

获取浏览器发送过来的数据

组成部分：

请求行 请求头 请求参数

操作请求行：

格式：请求方式 请求资源 协议/版本

常用方法：HttpSevletRequest

String getMethod(): 获取请求方式

String getRemoteAddr(): 获取请求的IP地址

String getContextPath()：在java中获取项目名称 (/day10)

了解 ：

StringBuffer getRequestURL() 带协议的完整路径

String getRequestURI() 从项目名到参数之前的内容

String getQueryString() 请求的所有参数的字符串

操作请求头：

格式：

key/value()value 可以是多个值

常用方法：

❤String getHeader(String key) :通过key获取指定的value(一 个)

了解：

Enumeration getHeaders(String name) 通过key获取指 定的value(多个)

Enumeration getHeaderNames() 获取所有的请求头名称

long getDateHeader(String name) 获取时间的请求头

int getIntHeader(String name) 获取整型的请求头

重要的请求头：

user-agent:浏览器内核信息

referer: 从哪个页面而来 防盗链

操作请求参数：

username=tom&password=123&hobby=drinhk&hobby=sleep

常用方法：

String getParameter(String key) :获取一个

String[] getParameterValues(String key) :通过一个key获取多个值

Map<String,String[]> getParameterMap() :获取所有的采纳数名称 和值

参数文件名称中文乱码：

对于get请求：参数追加到地址栏的时候会使用utf-8编码，但是服 务器（tomcat7）接收到请求之后使用iso-8859-1解码，所以乱码了

对于post请求：参数放在请求体中，服务器获取请求体的时候使用 iso-8859-1解码，也会乱码

方法1.通用的方法：

new String(参数.getBytes("iso-8859-1"),"utf-8");

byte[] b =str.getBytes("iso-8859-1") 把字符串变成指定字符集的数组

然后把数组拼成字符串：

new String(b,"utf-8")

方法2.

String filename\_ = new String(filename.getBytes("gbk"),"iso-8859-1");

针对于post请求来说，只需要将请求流的比编码设置成utf-8即可（上 面那种需要一个一个设置 参数太多不好设）

request.setCharacterEncoding("utf-8")

扩展

URLEncoder.encode(s,"utf-8")

URLDecoder.decode(s,"iso-8859-1")

29. map

获取keySet 或者 valueSet 键值对没有对应关系

map.keySet()

map.valueSet()

通过key获取value:

map.get(key) 值可能有多个

通过map.Entry 获得的是键值对

entrys = map.entrySet()

entry.getKey() entry.getValue()

---可以放入迭代器中

Map<String, String[]> map\_ = request.getParameterMap();

Iterator<Entry<String, String[]>> it = map.entrySet().iterator();

while(it.hasNext()){

Entry<String, String[]> entry = it.next();

System.out.println(entry.getKey()+" "+Arrays.toString(entry.getValue()));

}

注意：数组的话输出 Arrays.toString(Object[] a) 不然输出哈希码 乱码

30. 下载文件中文乱码的问题

超链接下载 filename得到的是乱码 所以找不到输入流 ，会出现空指针的问题

中文名称的文件名下载会出现名称问题，对于一般的浏览器需要提供 文件名称的 utf-8编码，火狐需要base64编码

31. 案例-用户注册

1.数据库和表

2.页面（表单）

给每字段添加name属性

还需修改表单提交的路径 post

3. 表单提交到servlet

4. 操作：

接受数据 封装成一个user

调用UserService完成保存操作 int regist(User user)

判断结果是否符合预期

int=1 ≠1 查看是否成功

相应的提示信息在另一个servlet 展示-----MsgServlet

5. UserService

调用dao

6. UserDao 通过dbutils在数据库插入记录

32. 封装对象

apache提供的一个工具类

BeanUtils

封装数据

使用步骤：

1.jar包

2.调用BeanUtils.populate(Object bean,Map<> 参数);

Map<> 参数就是获取的所有参数 从页面传来，这样就不用一个一个获取然后封装，直接接受了所有的参数

33. 请求转发和重定向的区别

重定向发送两次请求，请求转发发送一次请求

重定向地址栏发生改变，请求转发不变

重定向是从浏览器发送，请求转发是服务器内部

重定向不存在request域对象，请求转发可以使用request域对象

重定向是response的方法，请求转发是request的方法

response.sendRedirect("/day10/loc2")

request.getRequestDispatcher("/msg").forward(request, response)

重定向可以使用站外资源， 请求转发不可以

1. jsp cookie session
2. 案例1-记录用户上次访问时间

技术：会话技术

Cookie

Jsp

////////////////////////////////////

jsp本质上是一个servlet 在html中嵌套java代码 运行在服务器端，处理请求

生成动态内容

对应的java clsss在tomcat的work目录下

后缀名 .jsp

执行流程：

1. 浏览器发送请求 ，访问jsp页面
2. 服务器接受请求 jspServlet 帮我们查找对应的jsp文件
3. 服务器将jsp页面翻译成java文件
4. Jvm会将java编译 成.class文件
5. 服务器运行.class文件，生成动态的内容
6. 将内容发送给服务器
7. 服务器组成响应信息，发送给浏览器
8. 浏览器接收数据 解析展示

jsp脚本：