# java总结:

abstract 抽象类

interface 接口

implements 实现接口

## 泛型：

泛型通配符T，E，K，V区别：

使用大写字母A,B,C,D......X,Y,Z定义的，就都是泛型，把T换成A也一样，这里T只是名字上的意义而已

？ 表示不确定的java类型，即任何一种类型

T (type) 表示具体的一个java类型

K V (key value) 分别代表java键值中的Key Value

E (element) 代表Element也就是元素的意思

public class Student<T1, T2> {

// 泛型不能使用在静态属性或者静态方法上：

// 因为静态属性和方法是从属于类，在编译的时候就确定了类型。所有不能用在静态属性和方法上

private T1 javaScope;// java成绩

private T2 oracleScope;// oracle成绩

【总结：泛型相当于传进来的一个类型，可以直接调用，例如上方的T1和T2直接被属性调用不用再自己定义属性的类型】

泛型接口：接口中泛型字母只能使用在方法中，不能使用在全局常量上

泛型方法：修饰符 < T > 返回类型 方法名(){}，泛型方法可以定义在普通类中，也可以定义在泛型类中

指定泛型的范围：extends 表示<=

//关闭流 extends:表示<=Closeable 也就是只能是Closeable接口的实现类

public static <T extends Closeable> void closeIO(T...io){

## String类中的方法:

方法主要由非static修饰

1.equals();重写Object中的方法,不再比较对象地址,而是比较字符串内容

2.equalsIgnoreCase();比较两个字符串忽略字符串大小写

3.startsWith(String prefix)判断字符串前缀以什么开始

4.endsWith(String suffix)判断字符串后缀以什么结束

5.length()判断字符串长度

6.isEmpty()判断字符串长度是否为空 是返回true 不为空返回false

7.charAt(int index)利用字符串下标返回字符串

8.indexOf(String str)返回String字符串的下标

9.indexOf(int ch)返回char字符类型的下标

10.substring(int beginIndex)截取字符串从指定索引到末尾

11.substring(int beginIndex,int endIndex)截取字符串从指定索引到指定索引处

12.split(String regex)拆分字符串传入什么参数就以什么参数进行拆分---传入的参数抽取出来剩下的字符进行拆分

注意：还可以传入正则表达式，然后切割出自己想要的字符串

13.toCharArray()把字符串变成单个的字符存到数组里面,并返回

14.String()把字节数组变成字符串

15.getBytes()将字符串变为字节数组

toUpperCase()把字母变成大写

toLowerCase()把字母变成小写

16.valueOf()把任何类型的东西变成字符串

## StringBuffer: StringBuilder 两个类方法功能一致

方法主要由非static修饰 需要模板

注意:需要把字符串写在StringBuffer或StringBuilder的构造方法里面,这样使用方法才有效!

输出的时候要输出对象才能出结果

17.append()字符串后追加序列

18.insert()插入字符串

19.setCharAt()更新指定下标的字符串

20.delate删除指定下标后所有的字符串

21.delateCharAt删除指定字符下标到指定字符下标的字符出串

## 基本数据类型之间的转换:

### 1.基本数据类型 转换 包装类

构造方法(基本数据类型 参数)

静态方法valueOf(基本数据类型 参数)

自动装箱

### 2.包装类 转换 基本数据类型

包装类的方法 xxxValue() xxx是指基本数据类型byte short等…

自动拆箱

### 3.基本数类型 转换 字符串

1.+””

2.String类中的静态方法valueOf(数据类型 参数)

### 4.字符串 转换 基本数据类

包装类的静态方法parseXxx(String s) 可以将对应的字符串转为对应的基本数据类型

注意 :Character类没有该方法的

### 5.包装类 转换 字符串

String类中的静态方法valueOf(数据类型 参数)

### 6.字符串 转换 包装类

包装类的 构造方法(String s)

包装类的静态方法 valueOf(String s)

装箱拆箱

Jdk1.5后 有了自动装箱和自动拆箱的技术.

装箱 : 可以直接将基本数据类型变为包装类对象

## Math:

方法主要由static修饰

abs(int a) // 绝对值

ceil(double a) // 向上取整

floor(double a) // 向下取整

pow(double a,double b) // 两个参数第二个参数是第一个参数的次方

random() // 随机数

round(float a) // 四舍五入

sqrt(double a) // 开平方

## Random:

方法主要由非static修饰

这个类要创建对象才能使用;

Random.next xxx(参数);

## Date:

方法主要由非static修饰

Date() 获取当前时间把当前时间变成一个对象

Date(long time) 改变Date对象的时间参数

getTime() 获得毫秒值

## SimpleDateFormat:

方法主要由非static修饰

String partten = "yyyy-MM-dd HH:mm:ss";

y:年 M:月 d:日 H:时 m:分 s:秒 S:毫秒 E 星期天数

它允许进行格式化（日期 -> 文本）、解析（文本 -> 日期）

把日期变成字符串: 创建SimpleDateFormat对象在构造方法中输入模板;

用SimpleDateFormat类里面的方法Format传入当前时间,接收日期字符串;

把日期字符串变成Date对象,用SimpleDateFormat里面的parse(要改变的字符串参数)方法改变,用Date类型接收;

字母 日期或时间元素 表示 示例

G Era 标志符 Text AD

y 年 Year 1996; 96

M 年中的月份 Month July; Jul; 07

w 年中的周数 Number 27

W 月份中的周数 Number 2

D 年中的天数 Number 189

d 月份中的天数 Number 10

F 月份中的星期 Number 2

E 星期中的天数 Text Tuesday; Tue

a Am/pm 标记 Text PM

H 一天中的小时数（0-23） Number 0

k 一天中的小时数（1-24） Number 24

K am/pm 中的小时数（0-11） Number 0

h am/pm 中的小时数（1-12） Number 12

m 小时中的分钟数 Number 30

s 分钟中的秒数 Number 55

S 毫秒数 Number 978

z 时区 General time zone Pacific Standard Time; PST; GMT-08:00

Z 时区 RFC 822 time zone -0800

Calendar:2019年11月22日15:42:49 星期五

Calendar c = Calendar.getInstance();获取Calendar对象;

System.out.println(c.get(Calendar.YEAR)+"年"); 年

System.out.println(c.get(Calendar.MONTH)+1); 月

System.out.println(c.get(Calendar.DAY\_OF\_MONTH));日

System.out.println(c.get(Calendar.HOUR));时

System.out.println(c.get(Calendar.MINUTE));分

System.out.println(c.get(Calendar.SECOND));秒

System.out.println(c.get(Calendar.DAY\_OF\_WEEK));星期

## 异常:

try{

}catch(异常类型 变量名){

}finally{ finally最后执行

}

getMessage()获得异常信息

getCause()获得主要原因

printStackTrace()输出错误信息

自定义异常

1.继承异常父类 (继承不同类型的异常父类,使用的效果是不同的)

继承的编译期异常,或者是Exception异常,那么在使用时就必须对该异常进行处理

继承的是运行时异常,那么就不强制处理

2.将错误信息通过构造方法中的super传递给父类

编译期异常 : ParseException解析异常 / FileNotFoundException?文件未找到

运行时异常 : ArithmeticException算术异常 ClassCastException类型转换异常 IndexOutOfBounsException越界异常

NullPointException 空指针异常

## 集合:

add();存储

addAll();合并集合

size();获取集合长度

isEmpty();判断集合是否为空

contains();判断是否包含某个元素

equals();比较集合是否相等

remove(集合的下标);移除指定参数

toArray(传入想要变成的数组类型和长度);将集合变成数组

Iterator<String> a = a.iterator();

while(a.hasNext()){

String b = a.next();

system.out.println(b);

}

add(插入下标,插入的东西);

String a = get(0); 通过下标找元素

int b = a.indexOf("你");b = 0; 通过元素找下标

a.ramove(0);根据下标移除元素并返回移除的元素

set(修改的下标,修改的内容);

a.subList(0,1);截取集合内容;

ArrayList<E> a = new ArrayList<>(); List有序允许重复 数组

LinkedList<E> a = new LinkedList<>(); List有序允许重复 链表

HashSet<E> a = new HashSet<>(); set 无序不允许重复

LinkedHashSet a= new LinkedHashSet<>(); set 有序不允许重复

TreeSet<String> a = new TreeSet<>(); set 有序不允许重复

Hashmap<String,Integer> a = new Hashmap<>();

get(key);

keySet();获取键 返回Set<String>类型的集合

values();获取值 返回Collection<Integer>的集合

entrySet();获取对象 返回Set<Entry<String,Integer>>类型的对象集合 getkey();获取对象键 getvalue();获取对象值

containskey();包含键

containsvalue();包含值

## 操作目录和文件创建和删除:

exists();判断当前路径和文件是否存在;

mkdirs();创建多个文件夹（目录）

createNewFile();创建文件

delete();删除文件或目录

renameTo(文件路径对象);路径一样改名字,路径不一样剪切并改名字

isFile();判断是不是文件

isHidden();判断是不是隐藏文件

isDirectory();判断是不是文件夹

getAbsolutePath();获取绝对路径

getPath();获取构造方法中的路径3

getName();文件名

lastModified();文件最后一次修改的时间

length();文件长度

list();返回当前路径下有多少个文件

listFiles();返回一个文件路径对象

作业:

使用new File("D:\\A\\B"); // A B是文件夹

1. 判断A B是否存在,如果不存在就创建之

2. 在B文件夹中创建 1.txt 2.java 3.png 4.jpg 文件

3. 算出B文件夹中有几个照片文件

4. B文件夹保留1.txt 文件,其余都删除

## IO流：

### 字节流:

FileInputStream字节输入流

FileOutputStream字节输出流

BufferedInputStream

BufferedOutputStream

read();

write();

close();

flush();

### 字符流:

FileReader字符输入流

FileWriter字符输出流

BufferedReader

BufferedWriter

read();

write();

close();

flush();

### 高效流：

字节高效流：

BufferedInputStream

BufferedOutputStream

字符高效流：

BufferedReader

BufferedWriter

支持逐行的读写

方法和原始流一样

### 转换流

特点：把字节流转换成字符流

InputStreamReader:转换输入流

OutputStreamWriter:转换输出流

### 序列化：把内存中对象的字节信息，持久化存储到外部存储设备（文件）中

### 反序列化：把外部存储设备中的的字节信息，读到内存中重构成对象

序列化流：ObjectoutputStream

方法：writeObject(传入要序列化的对象);

反序列化流：ObjectInputStream

方法：readObject();

正则表达式：用String中的matches()方法传入正则表达式，与字符串进行比较

## 线程

### 创建线程的四种方式

<1>继承Thread ，<2>实现 Runnable ，<3>实现Callable <v> , <4>创建线程池

### 线程休眠

解释： 即线程进入阻塞状态后并进入就绪状态 ，

进入休眠的方法：

join(); A线程调用B线程的join()方法，A线程等待B线程终止后再运行

sleep(mills);定义当前线程休眠多少毫秒

wait();永久性休眠，只能用notify();或notifyAll();方法唤醒

### 线程同步

（锁对象）synchronized

主要用于修饰 代码块 ，修饰方法(锁对象是(this))，修饰类（锁对象是字节码（.class）文件）

锁对象特点：唯一性

死锁：两个线程互相等待，占用系统内存资源

解决方式：要尽量避免程序出现死锁

### Tcp通信协议：

Tcp属于服务端和客户端的关系

ServerSocket（服务端通信协议对象），Socket（客户端通信协议对象）

ServerSocket调用accept()方法获取字节流对象进行通信

Socket直接调用方法就可以获取字节流对象

### Udp通信协议：

（服务端，客户端）和（服务端，客户端）的关系

DatagramSocket(通信协议对象) , DatagramPacket（数据对象）

DatagramSocket调用send()方法就可以发送通信信息

DatagramSocket 调用receive()方法就可以接收数据

注意：，要往发送和接收的方法中传入DatagramPacket（数据对象）进行传输数据，也就是传输数据要用数据对象进行传输

DatagramPacket各个参数的意义：接收一般用前三个参数，发送要用所有的参数

DatagramPacket（字节数组，从字节数组的什么位置开始一般都是0 ，传入的字节数组的长度，ip地址，端口）；

### 注意：

### 字节码转字符串： new( byte[] , 0 , 字节数组的有效长度或总长度) 返回值 String

### 字符串转字节数组： “字符串”.getbytes(); 返回值 byte[];

## HTML

标签包括:开始标签和结束标签

Table表格线乱跑问题

要让table的宽度固定我们可以给table元素设置table-layout:fixed;样式，但这样表格的宽度问题是解决了，但是里面的内容并不会自动换行。我们还需要给td设置word-break: break-all; word-wrap:break-word;即可完美解决。

常用的块级标签

<https://blog.csdn.net/weixin_30838873/article/details/96238701>

**Article 和 Section：文章 和 部分，分段区域**

**Header 和 Footer：**头部和底部

**Nav：**页面的导航菜单部分

**Figure 和 Figcaption：数字和图表**

**Aside：侧边栏**

## css样式重点笔记:

shadow 阴影

radius 边框圆角

video 视频

audio 音频

source 定义视频和音频

animation: 动画名称 时间 速度 循环;

steps() 间隔动画

@keyframes 定义动画样式

transform: rotateX(0deg) transformZ(0px);标签位置 旋转 位移

perspective 景深3D视图

transform-style:preserve-3d;设置子标签也有3d效果

perspective-origin: 设置视角,就是视线端点眼睛的位置,默认处于中间位置,视线 水平 第一个值水平视角,第二个垂直视角

background: linear-gradient(red, blue);线性渐变

background: repeating-linear-gradient(red, yellow 10%, green 20%);重复的线性渐变

background: radial-gradient(red, green, blue);径向渐变

background: repeating-radial-gradient(red, yellow 10%, green 15%);重复的径向渐变

background-repeat: no-repeat;背景图默认重复显示,可以取消重复显示

## form表单:

输入框提示:属性:placenolder

类型: text文本框

` password密码框

radio单选框

checkbox复选框

submit提交按钮

button按钮

email邮箱

reset重置

checked默认选中单选框

## js代码重点笔记:

window.onkeydown = function(event){} window表示浏览器窗口,onkeydown表示键盘按下 获取键盘的方法

获取键盘的参数.keyCode 表示键盘上的键码值来区分不同的按键

load(); 加载

play(); 播放,开始

alert(""); 输出代码

var 要找的标签名 =document.getElementsByTagName("要找的标签名字")[index 第几个标签用数字表示]; 找HTML标签的代码

标签名+ . +标签名+ . +标签名 = ""; 找到标签后给标签添加样式的代码

变量: 没有数据类型限制 一个变量可以被赋值为其他数据 基本可以不写 var

var a = 1;

类型 : 数字类型 : number : 包括小数和整数

字符串类型 string : 字符 和 字符串;单引号 双引号都可以 ; 注意s是小写

布尔类型 boolean : 布尔类型,两个值true和false

未定义 undefined : 未定义 当使用未声明的变量是,出现undefined

null对象初始值为空

## Javascript:js

### 1:变量:

数字类型:主要有 number数字类型 和 string字符串类型 和 boolean 布尔类型 3种

10/3 结果是 : 3.33333333335

typeof()返回值的类型

### 2:运算符:基本和Java一样

==比较数字

===比较类型和数字

### 3:对象:

#### 数组对象:

创建数组的三种方式:长度可变,类型不固定

创建数组:var arr = new Array();

创建数组 : var arr2 = [];

创建数组: var arr3 = [1,2,3,4,5,6];

放值: arr[0] = 1;

arr[1] = 2;

alert(arr);结果为: 1 ,2

即使创建的数组有长度,但是可以继续添加元素

数组方法: pop();删除数组最后一个并返回删除的元素

push();向末尾添加一个或多个;

reverse();数组翻转

sort();数组排序

#### String对象:

length:属性! 字符串的长度

split();将字符串按照特殊字符切割成数组

substring(int index,int endIndex):提取字符串中两个指定的索引号之间的字符 包括开头不包括结尾

substr(int index,int endIndex):从起始索引号提取字符串中指定数目的字符

charAt(index i):返回索引字符

indexOf():返回字符的索引

toUpperCase();转大写

#### Date时间对象:

获取时间: var a = new Date();

getTime();获取当前时间毫秒值;

getFullyear();获取年

getmonth();获取月

getDate();获取日

getDay();获取当前星期

toLocalestring()获取本地时间字符串

#### Math数学类:

abs():绝对值

ceil():对数进行上舍入

floor():对数进行下舍入

random():0-1之间的随机数

round():四舍五入

### 4.函数:

function 函数名(){} 有名函数

function{} 匿名函数

function(参数1 ,参数2, 参数3){}

1.函数定义需要关键字function

2.函数名可写可不写,写了名字就可以通过名字调用

3.函数中没有关于返回值的定义,如果需要返回,直接在函数体中使用return值;返回即可

4.函数的参数列表中没有数据类型修饰,没有var修饰,可以写多个参数,之间用,隔开

5.调用有参数函数时,参数

6.匿名函数,需要配合事件直接使用;有名字可以通过函数名([值1,值2,值3]);调用;

7.调用有参数的方法时,实参可以多,但不能少

### 5.各种触发事件:

onclick();

### 6.事件的绑定:

1.事件和响应写在标签内

2.事件写在标签内,响应写在函数内

3.将事件和响应都写在标签外

### 7.弹框

js函数:window.location="路径";

alert();警告框

confirm()确认框 返回 true 和 false

prompt("提示信息")提示输入框 返回输入的值

@Webservlet("映射路径")

## DOM:

document.getElementById();

document.getElementsByClassName();

document.getElementsByTagName()[];

getElementsByName();返回标签名字集合

## Element:代表当前属性:

获得并设置标签属性:

className:修改class名字

innerHTML:设置或返回元素的内容(开闭标签之间的)

style:设置或返回标签样式

value:设置或返回标签的value的值

Element方法:

element.getAttribute(要获得的属性 参数);获得属性

element.removeAttribute(要移除的属性 参数);移除属性

element.setAttribute(要添加的属性 参数)添加属性

## 事件演示:

`

onclick按钮点击变红

confirm("是否删除");确认框 会返回ture或false

onfocus:获得焦点的事件

onblur:失去焦点的事件 [焦点就是指鼠标的光标]

onmouseover:鼠标悬浮的事件

onmouseout:鼠标离开的事件

## 正则表达式:RegExp

### 1.正常创建对象

var reg1 = new RegExp(正则模板);

var reg2 = /正则模板/;

#### 正则模板:

[]括号内,写正则的模板数据

{}括号内,写正则的模板数据的长度

[0-9]

[A-Z]

[a-z]

[A-z]

\d 代表数据

\D 非数字

\w 查找单词字符

\W 查找非单词字符

\s 查找空白字符

\S 查找非空白字符

n+ 出现至少一次或多次

n\* 出现0次或多次

n? 出现0次或1次

{5} 出现5

{2,8} 2到8次

向输入框中输入数据,然后输入完.在输入框后面出现

绿色"符合要求"

红色"不符合要求"

实现:

给输入框绑定 失去焦点事件

判断输入框中的值得模板是不是 指定模板 ==>正则模板

^:比对前面

$:比对后面

[0-9]只要出现字符串中出现一个就算对,不管几个都算对,不管有没有别的字符

^[0-9]$自己的数据 只能出现1个不能出现第二个0-9的数字才会匹配成功

{1,}最低一个不限制长度

[\u4E00-\u9FA5]:代表中文

创建var a = new RegExp("正则表达式");对象,括号里写正则表达式

js的text()方法相当于Java中的equals()方法;

## MySQL:安装

IP:确定电脑 端口:确定软件程序

### ` MySQL:卸载

卸载控制面板

卸载注册表

卸载安装路径的文件

卸载服务

### MySQL:使用

数据库:Database

环境准备:

安装mysql

启动mysql服务

配镜环境变量

进入MySQL:

win+r 输入cmd

进入数据库:mysql -uroot -p123456

查询端口号: show global variables like 'port';

退出数据库: exit;

## DDL -库/表的操作:

创建库: create database 要创建的数据库名字;

查询库: show databases;查询全部库

修改库的编码格式: alter database 已创建的库名 character set utf8;

查询库的全局编码格式:show variables like 'character%';

删除库: drop database 库名:

创建表顺序:

先选中库

use 库名;

### 创建表:

create table 表名(

字段1 数据类型(长度), --每个字段后用逗号隔开

字段2 数据类型 (长度),

字段3 数据类型 (长度) --最后一个字段不用写逗号

);

### 查询表:

show tables;

### 查看表的结构:

desc 表名;

### 修改表:

alter table 表名MODIFY 字段 数据类型(长度);

### 添加字段:

alter table 表名 add 字段名 数据类型(长度);

### 删除字段:

alter table 表名 drop 字段名;

### 修改表名:

RENAME TABLE 旧表名 TO 新表名;

rename table 旧表名 to 新表名;

### 删除表:

drop table 表名;

### 面试 :

drop truncate 和 delete有什么区别

drop 是直接将表删除

truncate 将表直接粉碎,然后创建一个新表

delete 将表中的数据删除,但是会有垃圾存留.

## mysql数据类型:

int：整型

double：浮点型，例如double(5,2)表示最多5位，其中必须有2位小数，即最大值为999.99；

char：固定长度字符串类型； char(10) 'aaa ' 占10位

varchar：可变长度字符串类型； varchar(10) 'aaa' 占3位`.

text：字符串类型,存大文本；

blob：字节类型,适合图片视频等字节文件；

date：日期类型，格式为：yyyy-MM-dd；

time：时间类型，格式为：hh:mm:ss

timestamp：时间戳类型 yyyy-MM-dd hh:mm:ss 会自动赋值

https://www.cnblogs.com/mxwz/p/7520309.html

datetime:日期时间类型 yyyy-MM-dd hh:mm:ss

## mysql : DML - 数据操作语言 -增删改

DML - 数据操作 - 数据增删改

增 - 插入

语法: insert into 表名(字段1,字段2,.....) values(值1,值2,....);

1.表名后的字段需要写在()内

2.表名后的字段没有要求与表中的数量一致,不要求与原表顺序一致

3.表名后的字段必须是表中的字段

4.values 后的值需要写在()内

5.values 后的值的数据类型要与前面字段的数据类型一致,并且顺序与前面的字段顺序一致

6.values 后的值的写法 : 数字直接写,字符串和日期类型需要用''括起来

-----------------------------------

### 增加: 增加数据

语法: insert into 表名(字段1,字段2,.....) valuses (值1,值2,....);此方法后面的括号必须和前面的括号的数据类型顺序保持一致;

insert into 表名 valuses (值1,值2,....);此方法必须和表的数据类型对照;

### 删除: 删除数据

delete from 表名; --删除全表的数据,谨慎!

delere from 表名 where 字段 关系运算符 值;

isnull判断是否为空

<>也代表不等于

日期也可以比大小

### 改变: 改变数据

update 表名 set 字段 = 值,字段 = 值,字段 = 值 where id = 2;可以修改一行中多个字段的值;

update 表名 set 要修改的一行中多个字段下的值 where id = 2;可以修改一行中多个字段的值;

where 后面代表条件语句;

更新数据时做加减乘除运算:

update stu set age = age+3 where sid = 1002;

update test set birthday = birthday+STR\_TO\_DATE('0002-01-01','%Y-%m-%d');

## 约束: 约束字段

在创建表时指定列的约束条件,后续向表中插入数据时要遵循的约束

主键: 主键一般都是id;

创建表时设置主键:语法: 字段 primary key, primary key(要设置的主键 字段名 后面)

一般创建表是添加约束

具有唯一性,不能为空;

组合键两个一样才算重复

auto\_increment设置主键自增功能

unique:设置该字段下的值不能重复，唯一约束

not null:设置该列字段下的值不能为空

default :设置该字段下的默认值

修改列的同时可以给列的类型添加约束

Alter table 表名 change 旧表名 新表名 类型 约束字段

## 外键:

主表（即就是老师表），从表（即就是学生表）

给学生表添加一个外键：

示例：

Alter table student add constraint stid foreign key (sid) references teacher(tid)

语法：Alter table 从表表名 add constraint 外键名 foreign key(从表的主键字段) references 主表表名(主表主键字段);

删除时/更新时:

? RESTRICT：

如果想要删除父表(被引用的)的记录时，而在子表(引用别人字段的表)中有关联该父表的记录，则不允许删除父表中的记录；

? 在我们演示中, stu是父表,class是子表

? NO ACTION：

同 RESTRICT，也是首先先检查外键；

? CASCADE：

父表delete、update的时候，子表会delete、update掉关联记录；

SET NULL：

/

父表delete、update的时候，子表会将关联记录的外键字段所在`

注释: comment '代码里的注释!';

#### 删除外键约束：

alter table 从表表名 drop foreign key 从表的外键字段名

## DQL - 查询:

select 字段1,字段2,字段3,....form 表名;

select 字段1,字段2,字段3,....form 表名 where 判断语句;

and相当于Java中的 &&

or相当于java中的 ||

is null 判断是否为空

is not null 判断不为空

between…and 什么什么之间

in(set) not in(set) 范围

查询语句字符串拼接函数: concat("字符串1","字符串2","字符串3",......)

### ANY , ALL , IN, NOT IN 的区别:

解释：主要在子查询时，子表返回一个集合时使用：

where A IN (SELECT B from 表) ：从表中查到的B 有一bai个等于duA

where A NOT IN (SELECT B from 表) ：从表中查到的B 都等于A

a >ANY(SELECT B from 表):a比从表中查到的最zhi小的B

a >NOT ANY(SELECT B from 表):a比从表中查到的最大的B

a >ALL(SELECT B from 表):a比从表中查到的最大B都大

a >NOT ALL(SELECT B from 表):a比从表中查到的最小B都大

### IN、ALL、ANY、SOME的解释

注意：IN 和NOT IN 不用运算符号就可以使用，ANY（SOME） 和 ALL 有运算符号才能使用

1. IN:在范围内的值，只要有就true
2. ALL（&&）: 与子查询返回的**所有值**比较为true 则返回true

与子查询返回的集合所有值进行比较，比较都成功才返回true

1. ANY（||）:与子查询返回的任何值比较为true 则返回true，

ANY : 与子查询返回的集合所有值比较，有一个成功就返回true,

1. SOME:是ANY的别称,很少用
2. NOT IN : 不在范围内就返回true

### ALL、ANY操作符的特殊情况

1. ALL的特殊情况
   1. 如果table1为空表 则都为TRUE
   2. 与如果子查询中有NULL 最终父查询返回结果都是NULL
2. ANY的特殊情况

### 模糊查询:

select id,name from emp where name like '%j%';

select form where 字段 like 值

百分号代表前后内容 和位置有关系 在前代表前面的内容 在后面代表后面的内容

like '%三';结尾为三

like '三%';开头为三

like '%三%'; 匹配任意字符

有几个下划线代表几个字 和位置有关系 在前代表前面的内容 在后面代表后面的内容

like '\_三';

like '三\_';

like '\_三\_' ;

like '\_三\_\_';

### 排序查询:

select 字段 form 表名 order by 字段 (asc升序|desc降序)

select name form ui order by age asc,age desc;在升序的基础上,重复的再进行降序

### 聚合函数:

count()：计算指定列不为NULL的记录行数；(查询字段下不为空 的 数据个数) 可以用where过滤

max()：计算指定列的最大值，如果指定列是字符串类型，那么使用字符串排序运算；(最大值)

min()：计算指定列的最小值，如果指定列是字符串类型，那么使用字符串排序运算；(最小值)

sum()：计算指定列的数值和，如果指定列类型不是数值类型，那么计算结果为0；(成绩和)

avg()：计算指定列的平均值，如果指定列类型不是数值类型，那么计算结果为0；(平均值)

distinct(): 去重查询

聚合函数的结果是一行数据,所以聚合函数只能和聚合函数一起查询不能和字段一起查询;

### 去重查询:

distinct(字段)

select distinct(字段) from 表名;

select count(distinct(字段)) from 表名; 去重并查询去重后的数据个数

### 分组查询:

group by 字段; having 判断语句; 先分组再过滤

提示:

having 和where 一样都是过滤 一般在order by和 group by 后使用

## 分页 : limit 意思:限制

语法: 在查询语句最后书写 : limit 数字 , 数字; 注意:不带括号!!!

select 字段 from 表名 limit 下标,当前下标往下要截取行数;

limit 下标 ,下标;

limit 数据下标, 从左边设置的下标往下截取数据;

## 流程控制语句:

case [when then] [else end]--> 与Java中的switch case 类似 哑表: from dual;

if(三目运算符) 例如:select IF (1 > 5 , '男' , '女') ，不能有问号和冒号，数据库不支持

if null(值1,值2) 值1不为(空null)执行值1 , 值1为(空null)执行值2;

error ex

## 多表:

union all 上下拼接

union 去重 上下拼接

示例：select \* from 表名1，union all 表名2 where sid = tid；

建议:两个表的 字段数量,个数,数据类型 保持一致\

## 连接查询:

### 内连接查询:关键字:

inner join on

表1 inner join 表2 on 表1.字段=表2.字段;

select \* from 表1 inner join 表2 on 表1.字段=表2.字段;

select \* from 表1,表2 where 表1.字段=表2.字段;

### 外连接:

关键字: (left/right) outer join

1.左外连接:特点:即使不满足条件也保存左表的数据 左右拼接

select \* from 表名1 left outer join 表名2 on 表名1.字段=表名2.字段;

2.右外连接:特点:即使不满足条件也保存右表的数据

select \* from 表名1 right outer join 表名2 on 表名1.字段=表名2.字段;

3.全外连接：特点：即使两个表都不满足条件也都可以显示出来

select \* from 表名1 full outer join 表名2 on 表名1.字段=表名2.字段;

### 内连接和外连接的区别：

内连接，也被称为自然连接，只有两个表相匹配的行才能在结果集中出现。返回的结果集选取了两个表中所有相匹配的数据，舍弃了不匹配的数据。由于内连接是从结果表中删除与其他连接表中没有匹配的所有行，所以内连接可能会造成信息的丢失。

内连接是保证两个表中所有行都满足连接条件，而外连接则不然。

外连接不仅包含符合连接条件的行，还包含左表(左连接时)、右表(右连接时)或两个边接表(全外连接)中的所有数据行。SQL外连接共有三种类型：左外连接(关键字为LEFT OUTER JOIN)、右外连接(关键字为RIGHT OUTER JOIN)和全外连接(关键字为FULL OUTER JOIN)。外连接的用法和内连接一样，只是将INNER JOIN关键字替换为相应的外连接关键字即可。

内连接只显示符合连接条件的记录，外连接除了显示符合条件的记录外，还显示表中的记录，例如，如果使用右外连接，还显示右表中的记录。

原文链接：https://blog.csdn.net/dove\_knowledge/java/article/details/71420206

### 子查询:

1.嵌套查询 当条件 当表用

## java连接数据库:

jdbc查询主要步骤:

1.加载驱动

2.注册连接

3.获得执行语句对象

4.执行sql,获得结果集

5.遍历结果集

6.关流

1.加载驱动

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

2.注册连接

jdbc协议 + mysql协议 + IP协议 + mysql端口 + 数据库

jdbc:mysql//localhost:3306/user

加载mysql驱动 驱动路径: com.mysql.jdbc.Driver

连接数据库 输入用户名 和 密码

获得一个执行对象

输入执行语句

遍历输出数据

关闭连接数据库流 关闭 执行对象流 关闭执行语句流

next()方法 查看还有没有下一条数据

ClassforName("驱动文件路径");

DriverManager.getConnection(数据库路径地址,用户名,密码 );

createStatement();

executeQuery("sql语句");

while(a.next())

{

getInt("id");

}

// //1.加载驱动

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

// //2.注册连接 获得连接对象

// String url = "jdbc:mysql://localhost:3306/user";

// String user = "root";

// String password = "123456";

// Connection coon = DriverManager.getConnection(url,user,password);

// //3.获得执行语句对象

// Statement s = coon.createStatement();

// //4.执行sql语句 - 查询id为1的用户

// ResultSet rs = s.executeQuery("select \* from users where id > 1");

// //5.遍历

// while (rs.next()) {

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

// System.out.println("id:"+ a);

// }

// //6.关流

// rs.close();

// s.close();

// coon.close();

预处理: ps = prepareStatement

ps.setxxx(下标,要传入的变量)

## web:

C/S模型 客户端 和 服务器

B/S模型 浏览器 和 服务器

请求方法 :get 会缓存

什么是http？

HTTP，即超文本传输协议，是一种实现客户端和服务器之间通信的响应协议，它是用作客户端和服务器之间的请求。

客户端（浏览器）会向服务器提交HTTP请求；然后服务器向客户端返回响应；其中响应包含有关请求的状态信息，还可能包含请求的内容。

### HTTP的常用方法

1、get方法

GET方法用于使用给定的URI从给定服务器中检索信息，即从指定资源中请求数据。使用GET方法的请求应该只是检索数据，并且不应对数据产生其他影响。

在GET请求的URL中发送查询字符串（名称/值对），需要这样写：

1

/test/demo\_form.php?name1=value1&name2=value2

说明：

GET请求是可以缓存的，我们可以从浏览器历史记录中查找到GET请求，还可以把它收藏到书签中；且GET请求有长度限制，仅用于请求数据（不修改）。

注：因GET请求的不安全性，在处理敏感数据时，绝不可以使用GET请求。

2、post方法

POST方法用于将数据发送到服务器以创建或更新资源，它要求服务器确认请求中包含的内容作为由URI区分的Web资源的另一个下属。

POST请求永远不会被缓存，且对数据长度没有限制；我们无法从浏览器历史记录中查找到POST请求。

3、head方法

HEAD方法与GET方法相同，但没有响应体，仅传输状态行和标题部分。这对于恢复相应头部编写的元数据非常有用，而无需传输整个内容。

4、put方法

PUT方法用于将数据发送到服务器以创建或更新资源，它可以用上传的内容替换目标资源中的所有当前内容。

它会将包含的元素放在所提供的URI下，如果URI指示的是当前资源，则会被改变。如果URI未指示当前资源，则服务器可以使用该URI创建资源。

5、delete方法

DELETE方法用来删除指定的资源，它会删除URI给出的目标资源的所有当前内容。

6、connect方法

CONNECT方法用来建立到给定URI标识的服务器的隧道；它通过简单的TCP / IP隧道更改请求连接，通常实使用解码的HTTP代理来进行SSL编码的通信（HTTPS）。

7、options方法

OPTIONS方法用来描述了目标资源的通信选项，会返回服务器支持预定义URL的HTTP策略。

8、trace方法

TRACE方法用于沿着目标资源的路径执行消息环回测试；它回应收到的请求，以便客户可以看到中间服务器进行了哪些（假设任何）进度或增量。

### http状态码总结:

http状态返回代码 1xx（临时响应）

表示临时响应并需要请求者继续执行操作的状态代码。

http状态返回代码 代码 说明

100 （继续） 请求者应当继续提出请求。 服务器返回此代码表示已收到请求的第一部分，正在等待其余部分。

101 （切换协议） 请求者已要求服务器切换协议，服务器已确认并准备切换。

http状态返回代码 2xx （成功）

表示成功处理了请求的状态代码。

http状态返回代码 代码 说明

200 （成功） 服务器已成功处理了请求。 通常，这表示服务器提供了请求的网页。

201 （已创建） 请求成功并且服务器创建了新的资源。

202 （已接受） 服务器已接受请求，但尚未处理。

203 （非授权信息） 服务器已成功处理了请求，但返回的信息可能来自另一来源。

204 （无内容） 服务器成功处理了请求，但没有返回任何内容。

205 （重置内容） 服务器成功处理了请求，但没有返回任何内容。

206 （部分内容） 服务器成功处理了部分 GET 请求。

http状态返回代码 3xx （重定向）

表示要完成请求，需要进一步操作。 通常，这些状态代码用来重定向。

http状态返回代码 代码 说明

300 （多种选择） 针对请求，服务器可执行多种操作。 服务器可根据请求者 (user agent) 选择一项操作，或提供操作列表供请求者选择。

301 （永久移动） 请求的网页已永久移动到新位置。 服务器返回此响应（对 GET 或 HEAD 请求的响应）时，会自动将请求者转到新位置。

302 （临时移动） 服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。

303 （查看其他位置） 请求者应当对不同的位置使用单独的 GET 请求来检索响应时，服务器返回此代码。

304 （未修改） 自从上次请求后，请求的网页未修改过。 服务器返回此响应时，不会返回网页内容。

305 （使用代理） 请求者只能使用代理访问请求的网页。 如果服务器返回此响应，还表示请求者应使用代理。

307 （临时重定向） 服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。

http状态返回代码 4xx（请求错误）

这些状态代码表示请求可能出错，妨碍了服务器的处理。

http状态返回代码 代码 说明

400 （错误请求） 服务器不理解请求的语法。

401 （未授权） 请求要求身份验证。 对于需要登录的网页，服务器可能返回此响应。

403 （禁止） 服务器拒绝请求。

404 （未找到） 服务器找不到请求的网页。

405 （方法禁用） 禁用请求中指定的方法。

406 （不接受） 无法使用请求的内容特性响应请求的网页。

407 （需要代理授权） 此状态代码与 401（未授权）类似，但指定请求者应当授权使用代理。

408 （请求超时） 服务器等候请求时发生超时。

409 （冲突） 服务器在完成请求时发生冲突。 服务器必须在响应中包含有关冲突的信息。

410 （已删除） 如果请求的资源已永久删除，服务器就会返回此响应。

411 （需要有效长度） 服务器不接受不含有效内容长度标头字段的请求。

412 （未满足前提条件） 服务器未满足请求者在请求中设置的其中一个前提条件。

413 （请求实体过大） 服务器无法处理请求，因为请求实体过大，超出服务器的处理能力。

414 （请求的 URI 过长） 请求的 URI（通常为网址）过长，服务器无法处理。

415 （不支持的媒体类型） 请求的格式不受请求页面的支持。

416 （请求范围不符合要求） 如果页面无法提供请求的范围，则服务器会返回此状态代码。

417 （未满足期望值） 服务器未满足"期望"请求标头字段的要求。

http状态返回代码 5xx（服务器错误）

这些状态代码表示服务器在尝试处理请求时发生内部错误。 这些错误可能是服务器本身的错误，而不是请求出错。

http状态返回代码 代码 说明

500 （服务器内部错误） 服务器遇到错误，无法完成请求。

501 （尚未实施） 服务器不具备完成请求的功能。 例如，服务器无法识别请求方法时可能会返回此代码。

502 （错误网关） 服务器作为网关或代理，从上游服务器收到无效响应。

503 （服务不可用） 服务器目前无法使用（由于超载或停机维护）。 通常，这只是暂时状态。

504 （网关超时） 服务器作为网关或代理，但是没有及时从上游服务器收到请求。

505 （HTTP 版本不受支持） 服务器不支持请求中所用的 HTTP 协议版本。

一些常见的http状态返回代码为：

200 - 服务器成功返回网页

404 - 请求的网页不存在

503 - 服务不可用

## Web服务器种类:

Apache

Tomcat

Jetty

JBoss

Nginx

## tomcat:服务器

无报错请求无响应：

https://blog.csdn.net/silencespy/article/details/95316135?utm\_medium=distribute.pc\_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase

tomcat运行会自动找 Java的jre运行环境 所以要配置JRE\_HOME环境变量

startup 启动服务器

shutdown 关闭服务器

使用Eclipse 连接服务器

Window --> preference --> server--> Runtime Environment --> add 添加服务器路径

搜素Servers 创建一个可用的服务器

服务器被占用解决方法: win + R 输入 cmd

第一种:

netstat -ano|findstr “8080” 找到 运行 8080 端口的 进程号

tasklist|findstr 进程号 根据进程号找到进程名

taskkill /f /t /im 进程名 根据进程名关闭进程

第二种:

找到服务器的配置文件夹conf 找到配置文件夹下的配置文件server.xml

创建Java Web 项目 必须带web.xml文件 没有的话 用eclipse软件生成

安装服务器

连接eclipse

创建服务器

创建Java Web 项目

## Web开发:

开发中主要使用: Httpserlet

使用方法:继承 Httpserlet使用: doGet() doPost() doPut() doDelete()

页面发出什么请求就使用什么方法

get 请求有4kb的内存限制

post 请求没有内存限制

可以直接从路径中发出映射路径请求,也可以直接用html发出请求 action="请求的映射路径" method="请求的方法"

页面发出请求后由映射路径接收并找到Java类

请求中有:

行 包括 :请求的方法 请求的路径 请求的协议

getMethod(); 获得请求方式

getRequestURL(); 返回客户端发出请求时的完整URL。

getRequestURI(); 返回请求行中的资源名部分。

getContextPath(); 当前应用的虚拟目录吧

getprotocol(); 获得http协议

头 包括 :请求的设置

getHeader(String name) 根据头名称得到头信息值

getHeaderNames() 得到所有头信息name

getHeaders(String name) 根据头名称得到相同名称头信息值

正文 包括 :发送给服务器的数据内容

getParameter(name) 根据表单中name属性的值，获取value属性的值方法

getParameterValues（String name）专业为复选框取值提供的方法 一个没选,默认是null

请求转发跳转java类: 请求转发是是服务器行为

注意:是在当前项目下拼接路径然后转发

getRequestDispatcher("第二个java的映射路径"); 重新从配置文件中的映射路径找到第二个Java代码

forward(req,resp);把请求和响应数据也携带过去

注意:请求跳转的第二个页面也要配置映射和类路径!!!

向请求域存储数据: 在第一个页面进行存储数据

req.setAttribute(key,value); key是String类型 = name value是object类型 = value

获取请求域存储的数据 在第二个页面进行获取数据

req.getAttribute("key");参数根据请求的键,获得值

设置响应状态码Status resp.setStatus();

设置响应内容类型contenttype resp.setcontenttype();

// 设置响应正文 : 获得一个字符输出流

PrintWriter writer = resp.getWriter();

// 向客户端输出字符

writer.write("响应页面... ");

// 在进行响应之前,对编码格式进行设置,对客户端的响应头设置

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

req.setCharacterEncoding("utf-8");请求乱码时设置

// 响应的内容类型ContentType=text/html;charset=utf8

// 改成JavaScript,浏览器就会以js的语法解析响应的数据

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

Tomcat服务器默认是utf-8

重定向跳转HTML页面:重定向是服务器行为

1. 重定向是浏览器/客户端动作

\* 2. 可以访问外部资源

\* 3. 两次请求,地址栏改变

\* 4. 请求域中的数据不能共享

resp.sendRedirect("第二个页面的地址");

请求重定向servlet : java类页面:

resp.sendRedirect();

getRequestDispcher().forward();

setAttribute();在请求域中存储数据

getAttribute();获取请求域中的数据

resp.setStatus();响应 修改状态码

resp.setcontenttype(); 响应 修改内容类型

resp.getwriter();获取输出流

write()响应页面;

setCharacterEncoding();设置请求的编码格式

sendRedirect();转换HTML页面

转发HTML页面:

jsp文件必须运行在服务器上才能展现效果

jsp文件可以写HTML标签和用<% %>写java代码

jsp内置对象: request

out//相当于java的输出语句

路径问题: 相对于当前上一层的路径

写绝对路径保险一点

### 重定向：错误日志：

错误日志：重定向发出第二次get请求路径中文乱码

解决方案：对要发送的数据进行编码，不能对整个路径进行编码，不然路径编码后会请求错误，只能对要发送的数据进行编码；

注意：只能是数据，除数据外的 （ ？键 = ）都不能进行编码 ，不然取不出值

示例：

第一次重定向：

@Override  
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
  
 String name = "张亮";  
 String newName = URLEncoder.*encode*(name, "UTF-8");  
 System.*out*.println(newName);  
 resp.sendRedirect(req.getContextPath()+"/Test?data="+newName);  
}

第二次发出get请求；req接收数据不用进行解码，直接设置请求的编码格式为UTF-8即可接收成功！

@Override  
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
 req.setCharacterEncoding("UTF-8");  
 resp.setCharacterEncoding("UTF-8");  
  
 String data = req.getParameter("data");  
 System.*out*.println(data);  
 resp.getWriter().write(data);  
}

## jsp:

### 三个指令:

<%@ 指令名 属性="" 属性="" 属性=""%>

#### page指令属性:

import=“包”v 指定脚本元素中使用到的Java类

session=“true|false” 用于指定该JSP是否参与到会话中，默认值为true

buffer=“none|size” 用于指定out对象使用的缓冲区大小，8kb为默认值

errorPage=“url” 用于当JSP页面发送异常时，转向的错误页面

isErrorPage=“false” 指定页面是否是错误处理页面，默认值为false

contentType=“ctinfo” 指定用于响应JSP页面的MIME类型和编码

pageEncoding=“utf-8” 指定JSP页面使用的字符编码

isELIgnored=“false” 指定JSP页面是否忽略EL表达式，默认值为false

#### include指令:引入任何界面

格式:<!--[指令] 包含: --><%@ include file="页面路径"%>静态包含=静态合并页面

<!--[动作] 包含:--><jsp:include page="页面路径"%>动态包含=动态加载页面

#### taglib指令:经常用于导入JSTL标签库=第三方标签库

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%> prefix:是导入第三方标识的名字;

### 六个动作:

#### 1.<jsp:include > 动态包含

用于在当前页面中包含动态和静态的资源

<jsp:include page=“url” > </jsp:include>

#### 2.<jsp:forward> 请求转发

将当前的请求转发给另外一个JSP页面或者Servlet

作用相当于RequestDispatcher的forward()方法

语法：

<jsp:forward page=“url”>

<jsp:param name=“name” value=“value”/>

</jsp:forward>

#### 3.<jsp:param> 设置请求参数

<jsp:param name=“name” value=“value”/>

#### 4.<jsp:useBean> 创建一个类的对象

<jsp:useBean id="name" class="package.class" />

#### 5.<jsp:setProperty> 给指定的对象属性赋值

<jsp:useBean id="myName" ... >

<jsp:setProperty name="myName" property="someProperty"/>

</jsp:useBean>

name属性表示要设置属性的是哪个Bean。

property属性表示要设置哪个属性

#### 6.<jsp:getProperty> 取出指定对象的属性值

<jsp:getProperty name="myName" property="someProperty" />

name 要检索的Bean属性名称。Bean必须已定义。

property 表示要提取Bean属性的值

### 九大内置对象:

不用创建,直接在页面中(jsp)使用

pageContext

request

session

application

response

exception

page

config

out

### EL表达式:

属性导航: 往属性里面存对象然后用EL表达式取出 (对象.属性)返回值是在请求域

算数运算:可以直接拿值进行运算

三目运算:可以使用三目运算符

集合: 可以取出集合

### JSTL:第三方标签库:主要用于jsp文件中

先在项目中的WebContent/WEB-INF/lib下导入jar包

在需要使用JSTL的JSP页面中,然后使用taglib命令导入标签库.

最后再写对应的标签

<c:set var="password"value="666"></c:set> 设置变量 往请求域中添加键和值相当于:req.SetAttribute();

<c:out value=""></c:out>输出 value是要是要输出的东西 ,可以设置默认值

<c:if test="${判断语句}"></c:if>判断为true是才会展现标签中的内容

<c:choose> 选择语句:相当于java的 switch case 语句

<c:when test="${判断语句}"></c:when>判断语句成功是展现标签内容

<c:when test="${判断语句}"></c:when>判断语句成功是展现标签内容

<c:otherwise></c:otherwise>都不满足时展现这个标签的内容

<c:choose>

<c:forEach var="i" items="${要遍历的集合 名字}">${i}</c:forEach> 循环/迭代

<c:forEach begin="0" end="10" step="2" var="i">&{i}</c:forEach>

<c:url value="要发出的请求路径"> 标签会自动补全路径

js函数:window.location="路径";

alert();警告框

confirm()确认框 返回 true 和 false

prompt("提示信息")提示输入框 返回输入的值

@Webservlet("映射路径")

### 四大域:

请求域-session域-pageContext域-application域:

请求域: setAttribute("","");存值 特点:转发一次,且只能共享一次

getAttribute("");取值

session域: req.session().setAttribute("",""); 存值 浏览器不关,能一直共享,跳转和转发都能收取到数据;

req.session().getAttribute(""); 取值

销毁session域中的数据: session.invalidate();

是把session内的所有属性 都清除，

session.removeAttribute("uiUsers")；

是清除session中的 "uiUsers" 属性，就是清除一个特定的属性

pageContext:page域,只在当前页面有效 存取有三个参数可以指定往其他域中存值;

pageContext.setAttribute();存

pageContext.getAttribute();取

Application:应用域,服务器不关就能一直共享

application.setAttribute();存

application.getAttribute();取

四大域优先级关系:pageContext > Request > session > application

路径问题: 请求转发是服务器行为

重定向是浏览器行为

在页面上加扛是从http://localhost:8080/根路径开始的 少一层项目名路径:例:dengluzhuce 不加杠是相对路径

在代码里加杠是从项目名根路径开始的 不加杠是相对路径 注意:相对路径等于是少写一个根路径

### cookie会话:

cookie是由服务器创建的

设置cookie方法: 创建cookie对象, Cookie cookie = new Cookie("键","值");

resp.addCookie(cookie);响应给浏览器设置Cookie

从请求中获取cookie:

Cookie[] cookies = req.Cookies();

if(cookies != null){

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

System.out.printIn("cookies[i].getName()+"="+cookies[i].getValue());

}

}else{

System.out.printIn("浏览器没有任何cookie");

}

设置cookie的保存时间: 设置保存时间时 注意:要在响应给浏览器设置cookie时设置cookie的保存时间;

默认是不保存;即是 -1;

cookie.setMaxAge(要设置的秒数);

映射路径写法:/ /\* \*.do /user/\*.do

### filter:拦截器

映射注解写法: Webfilter("映射路径") Webfilter({"/1","/2","/3"})拦截多个

映射xml写法: <filter-mapping>

<filter-name></filter-name>

<url-pattern></url-pattern>

</filter-mapping>

<filter>

<filter-name></filter-name>

<filter-class></filter-class>

</filter>

拦截语法: 实现;filter接口

重写doFilter(请求,响应,放行链)方法

chin.doFilter(请求,响应);将此处的请求和响应放行的方法

注意:

WEB-INF 下的资源属于服务器 , 只能用java代码的请求转发进行访问

## 日志框架:log4j

1.导入日志jar包

2.在java文件src路径下创建日志的配置文件 配置文件的后缀为: .properties

有四个级别:logger级别: DEBUG、INFO、WRNING、ERROR 小写: debug, info, wrning, error

### 输出DEBUG 级别以上的日志到=E://logs/error.log ###

#打印DEBUG级别的日志，有B和R两个用户

#是将为 debug 等级的日志，输出到 B 和 R 两个目的地

log4j.rootCategory=DEBUG,B,R

#将日志打印到控制台

log4j.appender.R=org.apache.log4j.ConsoleAppender

#模板对象  
 log4j.appender.R.layout=org.apache.log4j.PatternLayout

#模板  
 log4j.appender.R.layout.ConversionPattern=%-d{yyyy-MM-dd HH:mm:ss} [ %t:%r ] - [ %p ] %m%n

#打印的日志的驱动文件路径和方式

log4j.appender.B = org.apache.log4j.DailyRollingFileAppender

#打印的位置

log4j.appender.B.File = E://log4j\_logs/log.log

#是否追加打印的日志

log4j.appender.B.Append = true

#打印的级别

log4j.appender.B.Threshold = DEBUG

#打印的模板对象

log4j.appender.B.layout = org.apache.log4j.PatternLayout

#打印的模板

log4j.appender.B.layout.ConversionPattern = %-d{yyyy-MM-dd HH:mm:ss} [ %t:%r ] - [ %p ] %m%n

3.在java代码中创建对象

static Logger logger = Logger.getLogger(当前类名.class.getName());

4.有四种打印日志的方法:logger.debug(object);

logger.info(object);

logger.warn(object);

logger.error(object);

## c3p0的数据库连接池:

<c3p0-config>

<!--默认配置-->

<default-config>

<!--initialPoolSize：初始化时获取5个连接，取值应在minPoolSize与maxPoolSize之间。 -->

<property name="initialPoolSize">50</property>

<!--maxIdleTime：最大空闲时间,60秒内未使用则连接被丢弃。若为0则永不丢弃。-->

<property name="maxIdleTime">60</property>

<!--最大连接100个 -->

<property name="maxPoolSize">100</property>

<!--最少连接10个 -->

<property name="minPoolSize">10</property>

<!--maxStatements：最大链接数。-->

<property name="maxStatements">200</property>

</default-config>

<!--配置连接池mysql-->

<named-config name="mysql">

<property name="driverClass">com.mysql.jdbc.Driver</property>

<property name="jdbcUrl">jdbc:mysql://localhost:3306/day27</property>

<property name="user">root</property>

<property name="password">123456</property>

<property name="initialPoolSize">5</property>

<property name="maxIdleTime">30</property>

<property name="maxPoolSize">100</property>

<property name="minPoolSize">10</property>

<property name="maxStatements">200</property>

</named-config>

</c3p0-config>

//通过标识名来创建相应连接池

static ComboPooledDataSource dataSource = new ComboPooledDataSource("mysql");

dataSource.getConnection();

关流时正常关流包括(connection)都是一样正常关流连接会自动返还给数据库连接池

## 数据库事务介绍:

### 1.事务命令:

开启事务：start transaction | begin

提交事务：commit

回滚事务：rollback

### 2.事务的特性:

原子性（Atomicity）：指事务的整个操作是一个整体，要么都成功,要么都失败

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。转账前和转账后的总金额不变。

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）：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。

### 3.锁机制

Innodb默认是行锁,但是在事务(增删改)中,没有使用到索引,那么系统会自动升级会表锁,

主键就是索引, 命中索引 就是在where条件的字段是索引字段

行锁:事务开始后,其他用户不能操作该行数据,

表锁:事务开始后,其他用户不能操作这张表

### 4.事务的隔离级别:

隔离分为三种问题:脏读, 不可重复读, 虚读

分为4个级别:

1、读未提交-READ UNCOMMITTED: 赃读、不可重复读、虚读都有可能发生。

2、读已提交-READ COMMITTED: 避免赃读。不可重复读、虚读都有可能发生。（oracle默认的）

3、可重复读-REPEATABLE READ:避免赃读、不可重复读。虚读有可能发生。（mysql默认）

4、串行化-SERIALIZABLE: 避免赃读、不可重复读、虚读。

SELECT @@TX\_ISOLATION;

SET GLOBAL TRANSACTION ISOLATION LEVEL read UNCOMMITTED;

select @@tx\_isolation;//查询当前的隔离级别

set global transaction isolation level read uncommitted;//修改当前的隔离级别

### 5.事务在java中的使用方法:

conn.setAutoCommit(false); // 相当于开启事务

conn.commit();//提交 相当于结束事务

conn.rollback();//回滚 当SQL语句抛出异常时运行回滚代码,一般写在catch里面

## 反射:

### 获取类的.class文件三种方式:

1.通过路径获得类的.class文件

Class<Driver> forName = (Class<Driver>) Class.forName("com.mysql.jdbc.Driver");

2.通过类名得到类的.class文件

Class<User> clazz = User.class;

3.通过类的对象获得类的.class文件

User user = new User();

Class<? extends User> clazz = user.getClass();

获取类的.class文件属性的方式:

注意: 有Declared的方法 是用来获取有权限的属性,构造方法,普通方法的,获取后要修改权限

### 1.属性

Field age1 = clazz1.getDeclaredField("age");//括号中输入要获取的属性

getdeclaredFields //返回包含所有属性的数组

getField //

getFields//

get()

set()

### 2.构造方法

getDeclaredconstructor(构造的参数列表的.class文件);

getDeclaredconstructors()//返回所有的构造方法的数组

getconstructor()

getconstructors()

newInstance()运行方法

### 3.普通方法

getDeclaredMethod(方法名,参数列表的.class文件);

getDeclaredMethods()//返回数组

invoke()运行方法

### 4.注解

先获取类,或属性,或构造方法,或普通方法 然后在获取他们身上的注解

getAonn

设置属性,构造方法或普通方法的访问权限

clazz1.setAccessible(true);

### 总结:

1.获取类的class文件

2.利用class的方法获取class文件中的属性,构造方法,普通方法对象

3.根据获取的对象,利用对象中的方法 操作属性,构造方法或普通方法

4.操作后输出

5.对于有权限的属性,构造方法或普通方法要修改权限才能使用

## @注解的讲解:

创建注解文件 注解文件关键字 @interface

元注解: target(Elementtype.TYPE)作用范围

Retention(RetentionPolicy.RUNTIME)什么状态下运行

格式:数据类型 属性名() default 0 默认值

写value属性加 "" 默认值 不写时不会报错

## spring框架:IOC创建对象 控制反转

IOC:方法必须有构造方法

1.导入5个jar包

2.写xml配置文件

3.ClassPathXmlApplicationContext app = new ClassPathXmlApplicationContext(path);

HouseDaoimpl bean = (HouseDaoimpl) app.getBean("houseDaoimpl");

还可以不用强转直接获取:HouseDaoimpl bean =app.getBean("houseDaoimpl",HouseDaoimpl.class);

bean.setA(2);//利用获取到的对象输出

作用域: bean标签的 scope 作用范围的属性

四个作用域: 1.singleton//单列 xml文件默认单列

2.prototype//多列

3.request //请求域

4.session //会话域

## Dl注入:

<bean>

### set注入:

<property name="属性名" value="要赋的值"></property>这个标签有三个属性 name名字 value值 ref引用的值

### 构造方法注入:

<constructor-arg index="第几个参数" name="构造参数名" value="传入构造参数的值"></constructor> 这个标签有五个属性 type参数类型 ref引用类型

### 集合注入:

<property>

<Array>

<value><value> 或 <ref bean=""/>

</Array>

<list></list>

<set></set>

<map>

<entry key="" value=""></entry> 基本数据类型 或<entry key-ref="" value-ref=""></entry>引用数据类型

</map>

</property>

</bean>

### 利用注解实现IOC:关于spring创建对象的注解

用于创建对象的注解 加上该注解,就认为该类由Spring托管

@Component - 组件

@Controller – 控制

@Service - 业务

@Repository – 数据

以上四个注解作用都一样,只不过建议在不同的层面使用不同的注解以示区分

在控制层即Servlet 类/接口上 应该使用@Controller

在业务层即Service类/接口上,应该使用@Service

在数据访问层即Dao类/接口上,应该使用@Repository

如果其他类,想要有Spring托管,可以加@Component

@Value("值")给基本数据类型赋值

@Autowired给引用数据类型赋值. 根据类型进行判断赋值

@Qualifier("spring容器bean标签的id值")

xml:扫描注解的标签<context:component-scan base-package="要扫描哪个包的注解的路径">

## AOP:

### 1.代理:

代理类实现invocationHandler接口

代理运行:

Exercise01 exercise01 = new Exercise01(new Daoimpl());

DaonewProxyInstance=(Dao) Proxy.newProxyInstance(Daoimpl.class.getClassLoader(), Daoimpl.class.getInterfaces(),exercise01);

newProxyInstance.myAdd();

### 2.半自动代理:

切面类-

Spring的aop的切面类需要使用以下几种接口

Spring-AOP切面类实现的接口

? 前置通知 org.springframework.aop.MethodBeforeAdvice

在目标方法执行前实施增强

? 后置通知 org.springframework.aop.AfterReturningAdvice

在目标方法执行后实施增强

? 环绕通知 org.aopalliance.intercept.MethodInterceptor

在目标方法执行前后实施增强

? 异常抛出通知 org.springframework.aop.ThrowsAdvice

在方法抛出异常后实施增强

? 引介通知 org.springframework.aop.IntroductionInterceptor

在目标类中添加一些新的方法和属性

配置文件:

<bean></bean> spring 托管目标类 类型:引用

<bean></bean> spring 托管切面类 类型:String

-------------------

spring容器xml文件生成代理对象:用<bean></bean>标签写代理对象.代理对象的路径写:org.springframework.aop.framework.ProxyFactoryBean

用<property>标签给代理对象传值:

1.切面类 name="interceptorNames"

2.目标类的接口 name="interfaces"

3.目标类 name="target"

运行时找xml文件中的代理对象直接输出即可 注意!!!:用接口类型接收,两个参数(xml的代理对象的id名,接口的class的文件)!!!

###### 注意：Aop代理会给类上的每个方法都加个前置后置通知

可以对“方法名“进行模糊调用代理通知方法：

org.springframework.aop.support.RegexpMethodPointcutAdvisor

此类有两个属性：advice：代理的类

pattern：模糊匹配相当于正则匹配

使用：把此类bean的id名字写进list的value的标签中的即可，代替代理通知的id

### 3.全自动代理:

接口同上:实现类同上:切面类同上

配置文件:

<bean id="plantimpl" class="Agency.Plantimpl"></bean>

<bean id="section" class="Agency.Section"></bean>

<!-- 全自动代理 -->

<!-- <aop:config>

<aop:advisor advice-ref="section" pointcut="execution(\* Agency.Plantimpl.name(..))"/>

</aop:config> -->

直接从string容器中获得目标实现类来运行输出

### 4.Aspectj实现AOP代理:

<!-- Aspectj实现AOP -->

<bean id="myService" class="Agency.MyService"></bean>

<aop:config>

<aop:aspect ref="myService">

<aop:pointcut expression="execution(\* Agency.Plantimpl.\*(..))" id="namer"/>

<aop:before method="MyBefore" pointcut-ref="namer"/>

<aop:around method="MyAround" pointcut-ref="namer"/> //环绕需要方法传参数, 类型:ProceedingjoinPoint

<aop:after method="MyAfter" pointcut-ref="namer"/>

<aop:after-returning method="myAfterReturn" pointcut-ref="namer" returning="ret"/>

</aop:aspect>

</aop:config>

直接从string容器中获得目标实现类来运行输出

### 5.注解实现AOP代理:

@Aspect : 声明该类是一个切面类

@Pointcut : 声明这是一个切入点 [该注解要放在方法上]

@Before : 前置通知注解[该注解要放在方法上]

@After: 后置通知注解[该注解要放在方法上]

@Around 环绕通知注解[该注解要放在方法上]

@AfterReturning后置返回通知注解[该注解要放在方法上]

扫描代理注解的标签标签里不需要写任何东西,只写标签就行<aop:aspectj-autoproxy></aop:aspectj-autoproxy>

Spring - JdbcTemplate:

SpringJdbcTemplate相当于是Spring封装的DBUtil.

SpringJdbcTemplate必须使用数据库连接池.

BasicDataSource dataSource = new BasicDataSource();//创建数据库类

//给数据库设置 驱动文件 路径 账号 密码

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

dataSource.setUrl("jdbc:mysql://localhost:3306/day27?useSSL=false");

dataSource.setUsername("root");

dataSource.setPassword("123456");

JdbcTemplate jdbcTemplate = new JdbcTemplate();//创建事务类

jdbcTemplate.setDataSource(dataSource);//把创建好的数据库类给事务类

//查询数据表的语句：User user = jdbcTemplate.queryForObject(sql2, new Object[] {2},new MyRowMapper());//利用事务类里面的方法传入SQL语句

//SQL语句, 问号预处理的数组对象, 将查询到的数据放进对象的包装类

//修改数据表的语句：jdbcTemplate.update(sql,问号预处理的sql对象可以写多个参数)

System.out.println(user);//输出结果对象

//存储数据的包装类

public class MyRowMapper implements RowMapper<User>{

@Override

public User mapRow(ResultSet rs, int rowNum) throws SQLException {

User user = new User();

user.setId(rs.getInt("id"));

user.setName(rs.getString("name"));

return user;

}

}

## Spring框架连接数据库:

所有需要创建对象,属性注入的都交给Spring 容器去做.

创建数据库连接池对象：<bean id="dataSource" class="数据库连接池对象的路径">

<property name="driverClass" value="com.mysql.jdbc.Driver"></property>

<property name="jdbcUrl" value="jdbc:mysql://localhoust:3306/jk"></property>

<property name="user" value="root"></property>

<property name="password" value="123456"></property>

</bean>

创建jdbcTemplate对象：<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">

<constructor-arg index="1" ref="dataSource"><constructor-arg>//注入数据库连接池

</bean>

## Spring事务管理：

配置事务管理器：需要注入配置好的数据库连接池：

<!-- 事务管理器 -->

<beanid="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="dataSource"></property>

</bean>

配置事务管理器的切面类：transaction-manager传入的事务管理器

<!-- 创建增强事务的切面 -->

<tx:advice id="txAdvice" transaction-manager="txManager">

<tx:attributes>

<!-- 事务的传播行为：propagation 有八种传播行为 解释: 当事务发生冲突时，解决方法

事务的隔离级别：isolation 四种隔离级别 解释：事务与事务之间的隔离 -->

<tx:method name="transaction" propagation="REQUIRED" isolation="DEFAULT"/>

<!-- 强制回滚:指定回滚的异常类型 -->

<tx:method name="\*" rollback-for="Exception"/>

</tx:attributes>

</tx:advice>

定义AOP代理：

<!-- 定义AOP Spring代理 -->

<aop:config>

<aop:advisor advice-ref="txAdvice" pointcut="execution(\* com.zhiyou100.voi(..))"/>

</aop:config>

## 注解实现事务管理：

## springMVC框架:

1.web.xml配置方法：

1.配置Spring监听器：作用：用来监听项目，项目一启动就加载Spring配置文件

<!-- 配置spring的监听器 -->

<listener>

<listener-class>org.springframework.web.context.request.RequestContextListener</listener-class>//监听请求上下文

</listener>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>//监听内容加载程序

</listener>

2.先配置Webcontext下的xml文件: 找到src下的SpringMVC.xml文件

<!-- springmvc的前端控制器 -->

<servlet-mapping>

<servlet-name>DispatcharServlet</servlet-name>

<url-pattern>\*.do</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>DispatcharServlet</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:Servlet/Springmvc.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<!-- 前端控制器只是拦截路径的作用，启动Spring配置文件还是要用<context-param>标签进行加载 -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:Servlet/Springmvc.xml</param-value>

</context-param>

3.加载Spring的applicationContext.xml文件：此配置可以加载spring的任何配置文件

<!--加载配置spring配置文件applicationContext.xml-->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

【classpath：和 classpath\*：的区别：不带星号的加载src下的class文件，带星号的加载jar包和src下的 class文件】

2.SpringMVC配置文件方法：

再在src下配置对象xml文件

前提要有实现Controller接口的类，然后在Spring配置文件中创建对象，要写name属性里写：找到该类的映射路径

1.配置映射器<bean class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"></bean>

2.配置适配器<bean class="org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter"></bean>

3.配置视图解析器<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<!--请求转发或重定向的路径前缀-->

<property name="prefix" value=""></property>

<!--/请求转发或重定向的路径后缀-->

<property name="suffix" value=".jsp"></property>

</bean>

4扫描Controller的注解<context:component-scan base-package="要扫描的注解的包名"></context:component-scan>

<mvc:annotation-driven></mvc:annotation-driven>//此标签可以代表映射器和适配器,但是在类上需要写注解和映射路径,包括请求的方法

3.java代码写法：

需要在类上添加注解：@Controller

@RequestMapping("/写映射路径")

需要在方法上添加注解：@RequestMapping("/写找到这个方法的映射路径")

另一种写法：@RequestMapping(value="/写找到这个方法的映射路径" method = RequestMethod.GET)接收什么请求的方法

1.方法参数名字必须和页面标签的name属性里定义的名字一样才能接收信息，

可以直接用任何参数，包括请求（HttpServletRequest）和响应参数(HttpServletResponse)

页面跳转:

需要返回 请求转发：（forward:路径） 或 重定向：（redirect:路径）

静态资源映射: 需要在spring的配置文件中写 <mvc:resources location="要放行的文件路径" mapping="要放行的文件路径/\*\*" >

【请求的数据可以直接从请求的路径中直接取出：

@RequestMapping("/redis/${id}")

public String name(@PathVariable("id")int id){

}】

## spring拦截器:

1.自定义类实现HandlerInterceptor接口: 继承HandlerInterceptorAdapter类可以只用写一个方法,用哪一个就写哪一个

2.在spring的配置文件中,配置拦截器

<!-- springmvc的拦截器 : 可以配置多个-->

<mvc:interceptors>

<mvc:interceptor>

<!-- /\*\* 拦截所有,且拦截的只是我们已经定义过的Controller的路径 -->

<mvc:mapping path="/\*\*"/>

<bean class="com.zhiyou100.interceptor.MyInterceptor1" />创建拦截器对象 会自动调用所以不用写id名称

</mvc:interceptor>

</mvc:interceptors>

注意: <mvc:exclude-mapping path="/user/login"/>排除不拦截的路径

3.上传文件:MultipartFil

## MyBatis:

### 1.在Resources下配置全局配置文件：

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<!-- 数据库环境 -->

<environments default="development">

<environment id="development">

<!-- 事务管理器 -->

<transactionManager type="JDBC" />

<!-- 使用连接池 -->

<dataSource type="POOLED">

<!-- 数据库配置 -->

<property name="driver" value="com.mysql.jdbc.Driver" />

<property name="url" value="jdbc:mysql://localhost:3306/day19" />

<property name="username" value="root" />

<property name="password" value="123456" />

</dataSource>

</environment>

</environments>

<!-- 加载映射文件 -->

<mappers>

<mapper resource="myBatis01/Text01.xml"/>

</mappers>

</configuration>

### 2.在接口下配置映射配置文件

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE mapper

PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="myBatis01/Text01.java">

<!--

id:要执行的方法名

parameterType:入参类型

resultType:返回类型

resultMap:

-->

<select id="name" parameterType="MyBatis02.User" resultType="MyBatis02.User" resultMap="">

select \* from style where id = ${age}

</select>

</mapper>

### 3.测试运行mybatis

//1. 加载Mybatis全局配置文件

Stringresource="mybatis-config.xml";

InputStream inputStream=Resources.getResourceAsStream(resource);

//2. 通过全局配置文件得到SqlSessionFactory

SqlSessionFactory sqlSessionFactory=newSqlSessionFactoryBuilder().build(inputStream);

//3. 通过SqlSessionFactory得到SqlSession

SqlSessionsqlSession=sqlSessionFactory.openSession();

//4. 通过SqlSession得到Mapper代理对象

UserMapper userMapper=sqlSession.getMapper(UserMapper.class);

//5. 通过代理对象执行sql

List<User>users=userMapper.findAllUser();

for(Useruser:users){

System.out.println(user);

}

增删改需要提交事务才能有效：session.commit(); 或 openSession(true);自动提交事务

### 4.parameterType入参类型：

#{} 给字符串类型参数赋值，会自动拼接引号 一般用来取指定值和字符串值

${} 传入什么值就取出什么值，不会添加引号 一般用来取对象

执行增删改时需要自己提交事务 即在执行方法后和在sqlsion关流前 写 ： Sqlsion.commit(); 提交事务即可

JUnit 前置通知/增强方法 ，在本类的所有类执行前自动执行的方法 但 需要添加 @Before 注解 才能是JUnit前置方法，不然和普通方法没区别

1.基本数据类型参数，#{} 大括号内部的属性名随意

2.对象参数，也就是多个参数组合起来传进来一个对象， #{} 大括号内部的值 是对象的属性名

3.map参数，#{} 大括号中的值 要与map中的key保持一致

5.requesttype 返回类型：

1.查询时标签内才有这个属性，增删改时标签内没有这个属性，但接口方法也可以写返回值 但只能写int返回值，

mybis会自动封装影响的行数的返回值并返回

2.基本数据返回 ：

3.对

6.requestmap 返回一个map集合：有requestMap属性后就不在使用requesttype属性

在xml文件中自己定义一个map集合，要和自己查询后的虚拟表的字段一致否则封装不成功，就会返回失败

使用场景：<1>用于返回的虚拟表字段和返回对象不一致时

<requestMap type="返回的类型即自己创建的引用类" id="自己定义的名字即用来调用的名字">

<id column="" property="" /> //会自动查找主键

<result column="" property="" />

<result column="" property="" />

<requestMap>

column ：虚拟表显示的字段 property ：返回的引用类的属性，和虚拟表对照的属性，也就是类中的属性名

<2>用于返回多表联查时

### 【一对一】 ：

即查询的两个表的数据

在一个类中定义另外一个类的属性，即 一个类多出一个属性这个属性代表另外一个类

requestMap写法：<requestMap type="返回的类型即自己创建的引用类" id="自己定义的名字即用来调用的名字">

<id column="" property="" /> //会自动查找主键

<result column="" property="" />

<result column="" property="" />

<association property="" javaType="">

<id column="" property="" />

<request column="" property="" />

<request column="" property="" />

<request column="" property="" />

</association>

property : 当前类中的属性名 javaType ：当前属性的类型

<requestMap>

### 【一对多】 ：

和一对一步骤差不多，主要区别 ：添加的不是对象，添加的是list集合属性

一对多标签 ：<collection property="" oftype="">

<id column="" property="" />

<request column="" property="" />

<request column="" property="" />

<request column="" property="" />

</association>

</collection>

property : 类中的属性名 oftype ：写集合中存储的数据类型

### 7.一级缓存：

是sqlSion级别的，在sqlsion中多次获取同一个Mapper对象时只会发出一次请求且查询一次，

第二次就会在sqlsion缓存中寻找查询过的Mapper对象

【两个不同的sqlSion发出同一种请求，会查询两次，即不使用一级缓存，也就是说一级缓存只在sqlsion下才会默认使用】

【注意： 执行增删改后就会清空缓存，同一个方法才会去sqlsion里面去查找缓存】

【警告注意：要用colse方法关闭sqlsion,不然写不进去缓存】

### 8.二级缓存：

是Mapper级别的，Mapper配置文件中的所有方法都只会查询一次，下次调用就会从缓存中查找

a）二级缓存需要配置即开启缓存 即就是在MyBatis全局配置文件中添加配置：

<settings>

<setting name="cacheEnabled" value="true">

</settings>

b）开启后要使用二级缓存：

1.<cache></cache> 只用把这个标签放在映射配置文件中即可

2.这个标签默认使用 org.apache.ibtis.cache.impl.PerpetualCache 实现二级缓存的类

3.也可这样写把类加进标签内部：<cache type="org.apache.ibtis.cache.impl.PerpetualCache"></cache>两种方式都可以

4.开启缓存后正常从sqlsion里调用对象即可，但使用的是二级缓存，不是一级缓存

5.二级缓存底层原理是map ，方法名，参数（参数要注意，参数不一样时会重新再查一次数据库），一样时才会调用缓存

6.开启二级缓存后不同的sqlsion也会命中缓存

7.执行增删改后，也会清理缓存

【警告要用colse方法关闭sqlsion,不然写不进去缓存】

【BUG : 返回的数据序列化

解决方法 ：返回的对象实现序列化类接口即可 Serializable 】

### 9.动态sql ：

1.<where></where> 相当于 sql语句中的 where 关键字

2.<if test="keyword != null and keyword != ''"></if> 相当于java中的if判断 test 相当于 if判断后的括号里面写判断条件，

这个标签主要写在<where>标签里面，没有&& 要用 and 来写 and == &&

if ：

test 是布尔表达式

keyword可以是某一个对象的属性名或者是Map集合的键，可以不用写#{} 直接取值

判断表达式 与操作用的是单词 and相当于Java中的

${} ,如果是从Map中取值，写Map的键

3.<forEach collection="" item=""></forEach> 相当于java中的for循环

collection : 要遍历的集合

item : 遍历得到的结果

open ：开始遍历拼接前的东西

close ：结束遍历后拼接的东西

separator ：遍历的东西之间的间隔符

4.

### 10.mybatis逆向工程：

作用：自动生成和数据库相关的类和文件：

1.创建java 项目

1.导两个jar包，数据库连接包（MySQL的jar包）和 逆向工程的包（mybatis-generator-core-1.3.2.jar）

2.需要两个文件 : generator.xml 文件 和 运行的代码文件 即java文件

3.运行java代码文件就可生成相关的类和文件

#### <1>java代码写法：

import org.mybatis.generator.api.MyBatisGenerator;

import org.mybatis.generator.config.xml.ConfigurationParser;

import org.mybatis.generator.internal.DefaultShellCallback;

import java.io.File;

import java.util.ArrayList;

import java.util.List;

public class GeneratorSqlmap {

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

List<String> warnings = new ArrayList<String>();

boolean overwrite = true;

File configFile = new File("src/generator.xml");

ConfigurationParser cp = new ConfigurationParser(warnings);

org.mybatis.generator.config.Configuration config = cp.parseConfiguration(configFile);

DefaultShellCallback callback = new DefaultShellCallback(overwrite);

MyBatisGenerator myBatisGenerator = new MyBatisGenerator(config, callback, warnings);

myBatisGenerator.generate(null);

}

}

#### <2>generator.xml文件写法：

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE generatorConfiguration

PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"

"http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">

<generatorConfiguration>

<context id="testTables" targetRuntime="MyBatis3">

<commentGenerator>

<!-- 是否去除自动生成的注释 true：是 ： false:否 -->

<property name="suppressAllComments" value="true" />

</commentGenerator>

<!--数据库连接的信息：驱动类、连接地址、用户名、密码 -->

<jdbcConnection driverClass="com.mysql.jdbc.Driver"

connectionURL="jdbc:mysql://localhost:3306/video" userId="root"

password="123456">

</jdbcConnection>

<!-- <jdbcConnection driverClass="oracle.jdbc.OracleDriver"

connectionURL="jdbc:oracle:thin:@127.0.0.1:1521:yycg"

userId="yycg"

password="yycg">

</jdbcConnection> -->

<!-- 默认false，把JDBC DECIMAL 和 NUMERIC 类型解析为 Integer，为 true时把JDBC DECIMAL 和

NUMERIC 类型解析为java.math.BigDecimal -->

<javaTypeResolver>

<property name="forceBigDecimals" value="false" />

</javaTypeResolver>

<!-- targetProject:生成MODEL类的位置 -->

<javaModelGenerator targetPackage="com.zhiyou100.model"

targetProject=".\src">

<!-- enableSubPackages:是否让schema作为包的后缀 -->

<property name="enableSubPackages" value="false" />

<!-- 从数据库返回的值被清理前后的空格 -->

<property name="trimStrings" value="true" />

</javaModelGenerator>

<!-- targetProject:mapper映射文件生成的位置 -->

<sqlMapGenerator targetPackage="com.zhiyou100.mapper"

targetProject=".\src">

<!-- enableSubPackages:是否让schema作为包的后缀 -->

<property name="enableSubPackages" value="false" />

</sqlMapGenerator>

<!-- targetPackage：mapper接口生成的位置 -->

<javaClientGenerator type="XMLMAPPER"

targetPackage="com.zhiyou100.mapper"

targetProject=".\src">

<!-- enableSubPackages:是否让schema作为包的后缀 -->

<property name="enableSubPackages" value="false" />

</javaClientGenerator>

<!-- 指定数据库表 ==>这样会为每个表多生成一个xxxExample类-->

<!-- <table tableName="items"></table>

<table tableName="orders"></table>

<table tableName="orderdetail"></table>

<table tableName="user"></table> -->

<!-- 指定数据库表 ==>这样就不会为每个表多生成一个xxxExample类-->

<table tableName="video"

enableCountByExample="false"

enableUpdateByExample="false"

enableDeleteByExample="false"

enableSelectByExample="false"

selectByExampleQueryId="false"></table>

<table tableName="course"

enableCountByExample="false"

enableUpdateByExample="false"

enableDeleteByExample="false"

enableSelectByExample="false"

selectByExampleQueryId="false"></table>

<table tableName="permission"

enableCountByExample="false"

enableUpdateByExample="false"

enableDeleteByExample="false"

enableSelectByExample="false"

selectByExampleQueryId="false"></table>

<table tableName="role"

enableCountByExample="false"

enableUpdateByExample="false"

enableDeleteByExample="false"

enableSelectByExample="false"

selectByExampleQueryId="false"></table>

<table tableName="role\_permission"

enableCountByExample="false"

enableUpdateByExample="false"

enableDeleteByExample="false"

enableSelectByExample="false"

selectByExampleQueryId="false"></table>

<table tableName="speaker"

enableCountByExample="false"

enableUpdateByExample="false"

enableDeleteByExample="false"

enableSelectByExample="false"

selectByExampleQueryId="false"></table>

<table tableName="subject"

enableCountByExample="false"

enableUpdateByExample="false"

enableDeleteByExample="false"

enableSelectByExample="false"

selectByExampleQueryId="false"></table>

<table tableName="user"

enableCountByExample="false"

enableUpdateByExample="false"

enableDeleteByExample="false"

enableSelectByExample="false"

selectByExampleQueryId="false"></table>

<table tableName="user\_role"

enableCountByExample="false"

enableUpdateByExample="false"

enableDeleteByExample="false"

enableSelectByExample="false"

selectByExampleQueryId="false"></table>

<table tableName="log"

enableCountByExample="false"

enableUpdateByExample="false"

enableDeleteByExample="false"

enableSelectByExample="false"

selectByExampleQueryId="false"></table>

</context>

</generatorConfiguration>

### Spring整合mybatis的全局配置文件要传人数据库连接池才有效：

<!-- =================== spring整合mybatis ====================== -->

<!-- 数据库连接池 -->

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">

<!-- 配置数据源信息-->

<property name="driverClass" value="com.mysql.jdbc.Driver"></property>

<property name="jdbcUrl" value="jdbc:mysql://localhost:3306/his"></property>

<property name="user" value="root"></property>

<property name="password" value="123456"></property>

</bean>

<!-- 配置SqlSessionFactory -->

<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">

<!-- 引入MyBatis全局配置文件 -->

<property name="configLocation" value="classpath:MyBatis.xml"></property>

<!-- 传入数据库连接池 -->

<property name="dataSource" ref="dataSource"></property>

</bean>

<!-- 让spring托管mapping扫描功能,即让spring完成 生产mapper代理 过程 -->

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">

<!-- 扫描器注入SqlSessionFactory对象 -->

<property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"></property>

<!-- 扫描自己定义的mapper包的路径 -->

<property name="basePackage" value="mapper"></property>

</bean>

### Spring整合mybatis后还要配置全局配置文件，只是不需要写mapper配置，spring会自动配置

<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE configuration  
PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  
"http://mybatis.org/dtd/mybatis-3-config.dtd">  
<!-- 上面的dtd是当前这个xml文件的约束,类似于spring的xsd约束 -->  
<configuration>  
 <!-- 此处的一些列配置: ssm整合后,都可以交给spring去配置 -->  
   
</configuration>

### 4.MyBatis的分页插件：

导入maven依赖：

<dependency>  
 <groupId>com.github.pagehelper</groupId>  
 <artifactId>pagehelper</artifactId>  
 <version>4.2.0</version>  
</dependency>

#### 修改被Spring托管的SqlSessionFactory对象

注入：<!--分页插件-->

<property name="plugins">

<array>

<bean class="com.github.pagehelper.PageHelper">

<property name="properties">

<value>

<!--使用的数据库类型-->

dialect=mysql

reasonable=true

supportMethodsArguments=true

params=count=countSql

autoRuntimeDialect=true

</value>

</property>

</bean>

</array>

</property>

学习网址：<https://blog.csdn.net/weixin_41501963/article/details/93599288>

<property name="plugins"></property>设置分页插件的标签 //理解即可不需要粘贴到代码上

#### java代码写法：

在类中使用@Autowired 注入 PageHelper 类

pageHelper.startPage(PageNo , 3 );//在查询全部数据之前传入 （当前页 ，每页多少条数据）

userService.findAll(Map);//查询全部数据

PageInfo<User> PageInfo = new PageInfo<>(lists);//创建pageinfo对象把查询的全部数据放进Pageinfo对象中

***注意***：用pageinfo对象中的方法返回分页数据

req.setAttribute("PageInfo",PageInfo)把返回的数据带到页面上

req.setAttribute("PageInfoList",PageInfo.getList()); //PageInfo.getList() 查询到的全部的页面数据

页面写法：用EL表达式取出PageInfo对象中的 当前页 ， 总页数 ，上一页 ，下一页 .....

## 登录验证码 ：

需要到阿里的服务器学习使用：

## maven: 解压即可使用

1.maven需要配置环境变量，MAVEN\_HOME, 使用命令验证是否配置成功 mvn -v

2.maven需要配置本地仓库指向

<localRepository>D:\apache-maven-3.5.0-bin\apache-maven-3.5.0</localRepository>

3.maven需要配置镜像：镜像的主要作用让加载文件的速度更快

<mirror>

<id>alimaven</id>

<name>aliyun maven</name>

<url>http://maven.aliyun.com/nexus/content/groups/public/</url>

指向阿里的中央仓库

<mirrorOf>central</mirrorOf>

</mirror>

5.初始化maven

mvn help:system

6.关联eclipse：

whindos -> preferences -> maven -> 在Installations添加本地maven库 和 在User Settings中设置settings.xml文件

7.创建maven项目：

设置项目版本号 project Facets

点击Build path 关联 Tomact 服务器

## jQery:

利用<script type="text/javascript" src="jQery的路径">标签引入jQery的jar包，标签内部不能再写js代码

再写一个新的<script>标签里面写jQery的代码：

<1> 选择器：

作用：用于选中页面的标签

id:选择器 ： $("#标签的id值")

元素选择器/标签名选择器 ：$("标签名")

标签的class属性选择器 ： $(".class名")

属性选择器 : 选择器[属性名] 即 input[style]

选择器[属性名=值] 即 input[value="1"]

选择器[属性名!=值] 即 input[value!="1"]

<2> 属性操作：

解释：利用选择器选中标签，再操作标签的属性

1） $("#user").attr("要获得哪个属性")//获得属性值

.attr("value(属性名)"，"123456(修改属性的值)")//修改单个属性的值

.attr({name:"zs",value:"123"})//修改多个属性的值

.removeAttr("要移除的属性名")

2）标签class属性操作：

$("#user").addClass("c1")

$("#user").addClass("c2")//再添加一个样式名

$("#user").removeClass("要移除的属性名")

class属性可以添加多个样式名

平常标签添加多个样式名写法 class="c1 c2"

<3> HTML代码/文本值操作：

$("#user").html();//获得开闭标签间的内容 .html("内容")

$("#user").text();//获得开闭标签间的文本内容 .text("内容")

$("#user").val();//获得标签内value属性值 .val("内容")

<4> 标签css样式设置：

$("#in1").css("background-color")//获得css属性的值

$("#in1").css("background-color","green")//设置css属性值

$("#in1").css({"width":"100px","height":"100px","color":"red"})//设置多个css样式

<5> jQery事件：

绑定事件： blur([fn])//失去焦点事件

change([fn])//域内容改变事件

click([fn])//点击事件

dblclick([fn])//双击事件

error([fn])//错误事件

focus([fn])//获得焦点事件

keydown([fn])//键盘按下事件

keypress([fn])//键盘按压事件

keyup([fn])//键盘弹起事件

mousedown([fn])//鼠标按下事件

mousemove([fn])//鼠标移动事件

mouseout([fn])//鼠标离开事件

mouseover([fn])//鼠标越过事件

scroll([fn])//滑轮滚动事件

submit([fn])//提交事件

演示：$("#user").blur(function(){

alert("事件触发成功！");

})

事件处理：bind(type,[data],fn)

one(type,[data],fn)

trigger(type,[data])

## <6> json语法格式:

数据在 名称:值 对中 "username":"张三丰"

? 数据由逗号分隔 "username":"张三丰","age":"128"

? 大括号保存JSON对象 {"username":"张三丰","age":"128"}

? 中括号保存数组 {"class":[{"JAVA":"102"},{"BigData":"104"},{"HTML":"107"}]}

// json对象

var jsonObj = {"name":"张三","age":18};

// 获得JSON对象中的值 --> 对象.key

// alert(jsonObj.name + " - " + jsonObj.age);

// json数组

jsonArr = {"class":[

{"java":"101"},

{"H5":"102"},

{"Python":"103"}

]

};

// alert(jsonArr.class); // 返回的是数组,其中存储三个json对象元素

// alert(jsonArr.class.length); // 返回的是数组长度3

// alert(jsonArr.class[0]); // 返回数组中的第一个json对象

alert(jsonArr.class[0].java); // 返回数组中的第一个json对象的value值

## Ajax:

var b = new XMLHttpRequest();//创建Ajax传输数据的对象

        b.open("post", "url", true);//设置  （请求方式，请求路径，是否异步）

        b.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");//设置  请求内容的类型

        b.send("name='李四'&sex='男'&age=12");//发送给服务器的参数 post请求时写在send方法里面，get请求时直接写在url路径后面即可

        b.onreadystatechange = function () {//存有处理服务器响应的函数，每当 readyState 改变时，onreadystatechange 函数就会被执行

            if (b.readyState == 4) {//readyState响应的状态码 0: 请求未初始化（代理被创建，但尚未调用 open() 方法）

              alert(b.responseText);// 1: 服务器连接已建立（open方法已经被调用）

            }                       // 2: 请求已接收（send方法已经被调用，并且头部和状态已经可获得）

        }                           // 3: 请求处理中（下载中，responseText 属性已经包含部分数据）

                                    // 4: 请求已完成，且响应已就绪（下载操作已完成）

<1> 语法：$.ajax({

url:"向服务器发出的请求映射路径",

type:"请求的方法（get/post）",

data:{"name":"zs","value":"123"},//相当于在请求的映射路径后拼接数据

datatype:"text",//预期服务器返回的数据类型

contentType:"application/x-www-form-urlencoded",发送给服务器数据的参数内容编码类型

success:function(data){ //success 成功时的回调函数

data:服务器返回的数据

textStatus:数据的返回状态

jQXHR:ajax引擎

}，

error:function(){ //error 失败时的回调函数

}

成功时的回调函数和失败时的回调函数都用于控制前端页面

});

注意：Ajax使用post请求发送json数据到后台,contentType不能使用application/json,

使用默认即可application/x-www-form-urlencoded .

java代码：java响应页面数据

resp.getWriter().write("{\"code\"：\"200\"}");

<2> Ajax提交整个表单数据：

serialize (); 将表单数据序列化成 name=key&name=key&…

serializeArray(); 将表单数据变成数组,数组的元素是一个个json对象

var ArrayAjax = user.serializeArray();

alert(ArrayAjax[0].value);//取出的值

页面提交数据方法：

1）演示:ajax发送手动拼接json数据,contentType指定application/x-www-form-urlencoded,后台也可以直接封装进对象

2）演示:ajax发送手动拼接json数据,contentType指定application/json,后台不能封装进对象

把数据封装进对象用 @RequestBody注解，并定义在要封装的参数对象上,

ajax的data中的json格式需要是string类型的,可以手动加””引号,或者是使用JSON.stringify,

格式化为string格式,不然请求报错

<3> 返回数据方法：

页面返回数据要用对象返回，

对象要有三个属性：private String code;//状态码

private String msg;//信息

private Object data;//要返回的数据

//将返回值以json格式返回{"code":"200","msg":"封装成功",{"id":"1","name":"张三"}};

想要将类以JSON格式返回给前端,需要在方法上使用@ResponseBody注解导入Jackson的包.

java方法的返回值类型必须是RequestObject类型

<4> 图片或文件上传：

Commons-io commons-fileupload jar包或者maven依赖

maven 依赖代码： <dependency>

<groupId>commons-fileupload</groupId>

<artifactId>commons-fileupload</artifactId>

<version>1.3.2</version>

</dependency>

<dependency>

<groupId>commons-io</groupId>

<artifactId>commons-io</artifactId>

<version>2.4</version>

</dependency>

Springmvc配置文件上传解析：

<!-- 文件上传解析 -->

<bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">

<!-- 限制文件上传的大小 : 单位是byte 字节 B -->

<property name="maxUploadSize" value="10000000"></property>

<!-- 编码格式 -->

<property name="defaultEncoding" value="UTF-8"></property>

</bean>

java代码写法：

@RequestMapping(value = "/upload", method = RequestMethod.POST)

@ResponseBody

public ResultObject upload(MultipartFile img, HttpServletRequest request) throws IOException {

// 1 获取服务器路径 - 最终要上传的目的地路径

String path = request.getServletContext().getRealPath("/images");

System.out.println("目的地文件路径 : " + path);

// 2 判断要上传的文件的路径是否存在 ?

File file = new File(path);

// 如果不存在,建立这个目录

if (!file.exists()) {

file.mkdirs();

}

// 3获取文件的名字

String fileName = img.getOriginalFilename();

System.out.println("文件名 : " + fileName); // justdoit.jpg

// 4 确定上传的路径 /tomcat/webapp/springmvc02/images xxx.jpg

File descFile = new File(file, fileName);

// 5.文件上传

FileUtils.writeByteArrayToFile(descFile, img.getBytes());

System.out.println("上传完成");

String url = "\\images\\" + fileName;

request.setAttribute("path", url);

return new ResultObject("200","上传成功",url);

}

## Echarts:

作用解释 ：用于在浏览器页面展示各种报表 图形 雷达图 散点图 热力图 等等 特别适宜用在大数据领域

【可以在官网上自行学习，五分钟即可上手 ：https://www.echartsjs.com/zh/index.html】

1.下载 Echarts.js 文件

2.引入Echarts.js文件

<script src="echarts.js"></s

cript>

3.创建一个DOM 准成展现报表，为Echarts 准备一个具备大小（宽高）的DOM

<div id="main" style="width:600px;height:400px;"></div>

## Requful:

Requful ：只是一种页面发送请求的代码风格而已

功能 请求方式 url

添加 POST /CRUD/user

删除 DELETE /CRUD/user

更新 PUT /CRUD/user

查询 GET /CRUD/user

总结：以请求方式确定服务器资源

## 导入/导出 Excel文件：POI

<1> 导入jar包：

<!-- POI -->

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId>poi</artifactId>

<version>3.15</version>

</dependency>

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId>poi-ooxml</artifactId>

<version>3.15</version>

</dependency>

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId>poi-ooxml-schemas</artifactId>

<version>3.15</version>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-collections4</artifactId>

<version>4.1</version>

</dependency>

<2> 注意事项： 如果说项目启动运行时,报出关于org.Apache.poi…包 ClassNotFoundException时,

说明 关于poi的jar未找到

解决 确定一下本地仓库有无该jar,或者是有无下载完整

如果没有下载完整,把没有下载完整删除,重新下载.

先将该jar包的依赖从pom中删除

将项目清空clean,

项目右键 mavne -> updata project

重新将该jar包的依赖粘贴进pom,保存.重新查看仓库

<3> // -------POI操作--------

// 1 创建Excel表 // 创建的是XSSFWorkbook 导出是xlsx

HSSFWorkbook wb = new HSSFWorkbook(); // 导出的是xls

// 2创建Sheet页面

HSSFSheet sheet = wb.createSheet("医生信息");

// 3在sheet页面中创建行 , 第一行是0

HSSFRow row = sheet.createRow(0);

// 4在行中创建列(单元格)

HSSFCell cell = row.createCell(0);

// 5设置内容

cell.setCellValue("医生ID");

// 6导出(固定到指定地方)

OutputStream os = new FileOutputStream("C:\\Users\\qiush\\Desktop\\1.xls");

// 执行写出

wb.write(os);

os.close();

System.out.println("完成");

}

## 用户权限框架 shiro:

1.导入jar包：

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-simple</artifactId>

<version>1.7.21</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>jcl-over-slf4j</artifactId>

<version>1.7.21</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-core</artifactId>

<version>1.2.2</version>

</dependency>

2.写shiro的配置文件：

Shiro的配置文件是ini格式 , 后缀是ini

Ini格式文件与properties文件比较像,都是key=value形式

配置文件格式：认证：

[users]

# 用户名=密码

zs=123456

ls=123456

3.java代码写法：

// 得到配置文件工厂

IniSecurityManagerFactory factory = new IniSecurityManagerFactory("classpath:shiro.ini");

// 通过工厂获得安全管理器

SecurityManager instance = factory.getInstance();

// 将SecurityUtils设置为全局属性

SecurityUtils.setSecurityManager(instance);

// 从SecurityUtils中获得主体

Subject subject = SecurityUtils.getSubject();

// 通过用户登录时输入的用户名密码

UsernamePasswordToken token = new UsernamePasswordToken("ls","123456");

try {

subject.login(token);

} catch (UnknownAccountException e) {

System.out.println("账号错误！");

} catch (IncorrectCredentialsException g) {

System.out.println("密码错误！");

}

System.out.println("认证成功！");

4.Realm域 认证：

<1> 因为ini文件中的内容是写死

的，所以要用域来代替，

但ini文件中必须要有：

# 声名一个自定义类

myRealm1=com.zhiyou100.muRealm1

# 指定SecurityManager使用Realm

SecurityManager.realms=myRealm1

<2> 创建Realm自定义域对象，继承AuthorizingRealm授权类，这个类中包括 认证和授权

doGetAuthorizationInfo()//这是一个授权方法

doGetAuthenticationInfo()//这是一个认证方法

获取用户名：String username = (String) token.getPrincipal();

获取密码：String password = new String((char[])token.getCredentials());

从数据库中查询并获取用户数据并对比，错误抛出异常

if (!"zs".equals(username)) {

throw new UnknownAccountException();

}else if (!"123456".equals(password)) {

throw new IncorrectCredentialsException();

}

用户名，密码都正确即认证成功，返回 用户名和密码

new SimpleAuthenticationInfo(username,password,getName());

5.授权

配置文件授权格式：

[users]

#用户名=密码，角色1，角色2，角色3

zs=123456,emp

ls=123456,emp,admin

[roles]

#角色=权限1，权限2，权限3

#权限可以写：user/add 也可以写：user：add

emp=user/add

admin=user/add,user/delete

emp=user:add,

admin=user:add,user:delete

[urls]

# anon允许匿名访问,注释一定放在上方

/login=anon

/logout=anon

/js/\*\*=anon

/css/\*\*=anon

/index.jsp=anon

# authc,是需要需要认证后才能访问

/user/list=authc

# /user/add路径,需要登录,且授予admin角色,并赋予user/add权限才可访问

/user/add=authc,roles[admin],perms["user/add"]

# /user/update路径,需要登录,且授予admin角色可访问

/user/update=authc,roles[admin]

Realm域授权：

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection arg0) {

// 获取用户名

String primaryPrincipal = (String) arg0.getPrimaryPrincipal();

// 根据用户名，查询数据库

String a = "zs";

// 创建集合，把从数据库中查询到的角色，添加进集合里面

ArrayList<String> jaose = new ArrayList<String>();

jaose.add("员工");

jaose.add("管理员");

// 创建集合，把从数据库中查询到的权限，添加进集合里面

ArrayList<String> quanxian = new ArrayList<String>();

quanxian.add("user/add");

quanxian.add("user/delete");

// 把角色集合和权限集合添加进SimpleAuthorizationInfo()对象里面，并返回

SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();

info.addRoles(jaose);

info.addStringPermissions(quanxian);

return info;

}

在测试页面利用[主体(subject)]判断是否有 角色，权限：

<1> 创建集合，存入要判断的角色

subject.hasRoles();//判断角色

subject.isPermitted();//判断权限

## 整合: shiro - web:

需要使用shiro的监听器,监听项目启动,启动后立即加载并初始化shiro的配置以及拦截器

<!-- 配置shiro监听器 -->

<listener>

<listener-class>org.apache.shiro.web.env.EnvironmentLoaderListener</listener-class>

</listener>

<!-- 加载并初始化shiro的配置文件 -->

<context-param>

<param-name>shiroConfigLocations</param-name>

<param-value>classpath:shiro-web.ini</param-value>

</context-param>

<filter>

<filter-name>ShiroFilter</filter-name>

<filter-class>org.apache.shiro.web.servlet.ShiroFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>ShiroFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

其他的和上面记得笔记一样！

## Spring 整合 Shiro :

1.导入jar包:

<!--shiro-->

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-core</artifactId>

<version>1.2.2</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-web</artifactId>

<version>1.2.2</version>

</dependency>

<!--shiro与Spring整合-->

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-spring</artifactId>

<version>1.2.5</version>

</dependency>

2.shiro的Spring配置文件：

<!--===================spring整合Shiro======================-->

<!--创建自定义Realm对象-->

<bean id="myRealm1" class="loginController.MyRealm1"></bean>

<!--创建SecurityManager,并注入自定义Realm-->

<bean id="securityManager"

class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">

<property name="realm">

<ref bean="myRealm1"></ref>

</property>

</bean>

<!--Shiro的Web过滤器id必!须!是!web.xml中配置的Shiro过滤器的filter-name-->

<bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">

<property name="securityManager" ref="securityManager"></property>

<!--登录的url:相当于ini文件中的authc.loginUrl=/login-->

<property name="loginUrl" value="/login.html"></property>

<!--认证授权失败跳转的错误页面:相当于是ini文件中

roles.unauthorizedUrl=/error.jspperms.unauthorizedUrl=/error.jsp-->

<property name="unauthorizedUrl" value="/login.html"></property>

<!--配置的拦截路径:相当于是ini文件中的[urls]配置-->

<property name="filterChainDefinitions">

<value>

<!--从上至下的逐一匹配-->

<!--静态资源放行-->

/assets/\*\*=anon

/Css/\*\*=anon

/front/\*\*=anon

/Images/\*\*=anon

/Js/\*\*=anon

<!--普通页面放行-->

/index.jsp=anon

/index=anon

/error.jsp=anon

/login.jsp=anon

/login.do=anon

/logout.do=logout

<!--首页需要认证-->

/registration/\*=authc

/user/index.do=authc

<!--该路径需要认证且必须有admin角色才可以访问,在Realm中进行授权-->

/user/add.do=authc,roles[admin]

<!--该路径需要认证且必须有/user/delete权限才可以访问,在Realm中进行

授权-->

/user/delete.do=authc,perms["/user/delete"]

<!--从上至下依次匹配路径,如果以上都没匹配成功,走最后一个/\*\*-->

/\*\*=authc

</value>

</property>

</bean>

web xml 添加：shiro的拦截器：

<!-- Shiro的拦截器 filter-name要与Spring中的ShiroFilterFactoryBean的id保持一致 -->

<filter-mapping>

<filter-name>shiroFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<filter>

<filter-name>shiroFilter</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

</filter>

### 3.shiro 加密：

jar包或者maven依赖：

Base64 编码/解码

SHA

MD5 不可逆的，加密后，无法再逆向得到原文

<1> 创建MD5对象：

String md5 = new Md5Hash(密码，盐值，循环加密的次数)；

下次登录认证后要把

returnnewSimpleAuthenticationInfo(user.getUserName(),user.getPassword(),byteSource,getName());返回

返回从数据库查询到用户名，密码，盐值，get.Name();

盐值要变成shiro需要的字节码格式返回，即：ByteSource byteSource = ByteSource.Util.bytes(盐值);

想要从shiro的token里得到用户对象，用户名就必须返回查询到的用户对象，想要得到字符串正常返回即可

获取主体中的对象：subject.getPrincipal(); 强转即可使用对象

/\*

\*已经告知Shiro,

\*明文密码:Principal

\*密文:user.getPassword()

\*盐值:byteSource

\*----

\*未知

\*加密方式:?

\*加密循环次数:?

\*----

\*所以需要重写加密解密的方法->凭证匹配器

\*指定加密次数以及加密方式

\*---

\*在于Spring整合以后,以下的方法其实可以写成Spring的Bean,在bean设置对应的属性

\*/

不用实现接口或继承类，可直接使用此方法：返回 加密方法，加密次数

//@Override

//public void setCredentialsMatcher(CredentialsMatcher credentialsMatcher){

//HashedCredentialsMatcher hashedCredentialsMatcher = new HashedCredentialsMatcher();

////设置加密方法

//hashedCredentialsMatcher.setHashAlgorithmName("MD5");

////设加密次数

//hashedCredentialsMatcher.setHashIterations(10);

//super.setCredentialsMatcher(hashedCredentialsMatcher);

//}

<!-- ================spring整合Shiro============== -->

<!-- 创建自定义Realm对象 凭证匹配器 -->

<bean id ="myRealm1" class="com.zhiyou100.shiro.myRealm1">

<property name="credntialsMatcher">

<bean class="org.apache.shiro.outhc.credential.HashedCredentialsMatcher">

<property name="hashAlgorithmName" value="MD5"></property>

<property name="hashIterations" value="10"></property>

<bean>

</property>

</bean>

## Linux常用命令：

常用命令：

ifconfig 查看ip地址

pwd 查看当前所在的目录

date 查看当前的时间

who 查看当前用户

last 查看最后一个登陆的用户

whoami 我是谁

clear 清屏

reboot 重启

halt 关机

hostname 查看主机名

常用文件系统操作

cd 目录名 进入某个目录下

cd .. 返回上一级目录

查看操作

ls 查看当前目录下的内容

ls -a (all) 查看当前目录下的所有内容(隐藏内容)

ls -l 查看更详细的信息

ls -a -l / ls -al /ll

创建目录(文件夹)

mkdir 目录名

mkdir -p 多级目录

删除目录

rm 文件名

rmdir 目录名 (删除的是目录，并且这个目录里面没有内容)

rm -rf 目录名(目录里面可以有内容，也可以删文件)

移动目录(文件)

mv 原来的文件 移动后的文件

复制目录(文件)

cp -r 原来的 之后的

文件操作

touch 文件名 (创建了一个空的文件)

cat 文件名 查看文件的内容

echo “内容”>文件名 (如果文件存在的话，会被覆盖)

echo “内容”>>文件名 (如果文件不存在，创建文件并写内容，如果文件存在，追加内容)

vi 是编辑命令

vi 跟一个不存在的文件名，会创建这个文件，并且编辑

如果vi 跟了一个存在的文件名，会打开并且编辑

Vi 文件名：

1. 进来之后就是一个一般模式，在这个模式下面，不能编辑，但是可以使用快捷键

2. 按 i 进入编辑模式(可以修改内容)

3. 按esc 退出编辑模式-又进入了一般模式

4. 退出：

：q! 退出但是不保存

：wq 退出并保存

vi 常用快捷键

在一般模式下才可以使用快捷键

gg到文件的第一行

G (大写G) 到最后一行 shift+g

dd 删除某一行

3dd 删除3行

5dd 删除5行

yy 复制

p 粘贴

:/ 查询的内容 确定

N 上一个 n 下一个

:set nu 显示行号

用户的基本操作

添加用户

useradd 用户名

(添加了这个用户之后不能直接使用，需要给这个用户设置密码)

passwd 用户名

切换用户

su 用户名

权限

常见的权限表示形式有：

-rw------- (600) 只有拥有者有读写权限。

-rw-r--r-- (644) 只有拥有者有读写权限；而属组用户和其他用户只有读权限。

-rwx------ (700) 只有拥有者有读、写、执行权限。

-rwxr-xr-x (755) 拥有者有读、写、执行权限；而属组用户和其他用户只有读、执行权限。

-rwx--x--x (711) 拥有者有读、写、执行权限；而属组用户和其他用户只有执行权限。

-rw-rw-rw- (666) 所有用户都有文件读、写权限。

-rwxrwxrwx (777) 所有用户都有读、写、执行权限。

1可执行

2可写

4可读

修改权限

+ 增加某个权限

- 去掉某个权限

chmod g+|-x(x是具体权限) 文件名 g group 组

chmod o+|-x 文件名 o other 其他用户

chmod u+|-x 文件名 u user当前用户 自己

chmod 777 文件名

chmod 600 文件名

查看当前进程的命令：

ps -aux //查看当前进行的进程

ps -aux|grep +进程英文名 //查看具体的某个进程

输入上述命令后 输入 top 命令可以实时更新查看进程状态，如果要退出这种状态 输入 ：q 命令，就可以退出了

如果要查看所有进程(包括当前进行的所有进程)，就输入命令 ps -|A 那么此时就会看到所有进程了

杀死进程：kill -9 [pid名字]

通过端口号查询进程：

netstat -p | grep 3306（端口号）

Linux 的软件安装目录是也是有讲究的，理解这一点，在对系统管理是有益的

/usr：系统级的目录，可以理解为C:/Windows/，/usr/lib理解为C:/Windows/System32。

/usr/local：用户级的程序目录，可以理解为C:/Progrem Files/。用户自己编译的软件默认会安装到这个目录下。

/opt：用户级的程序目录，可以理解为D:/Software，opt有可选的意思，这里可以用于放置第三方大型软件（或游戏），当你不需要时，直接rm -rf掉即可。在硬盘容量不够时，也可将/opt单独挂载到其他磁盘上使用。

源码放哪里？

/usr/src：系统级的源码目录。

/usr/local/src：用户级的源码目录。

总结：

<1>用 Xshell 6 控制 虚拟机

用 xftp 6 传输文件

<2>创建：

mkdir 目录名

mkdir -p 多级目录

vi 文件名 //编辑文件，没有这个文件，创建这个文件

<3>删除：

rm 文件名

rmdir 目录名 (删除的是目录，并且这个目录里面没有内容)

rm -rf 目录名(目录里面可以有内容，也可以删文件)

<3>加载vim插件

yum install vim

tar.gz : 压缩命令： tar -zcvf 压缩文件名 .tar.gz 被压缩文件名

解压命令： tar -zxvf 压缩文件名.tar.gz

## Linux jdk 安装：

【注意：Linux系统安装任何软件都需要安装到 /usr/local 路径下】

1.上传文件 到 /usr/myfile 下

2.解压

3.安装 yum install vim 插件

4.配置环境变量：环境变量文件路径：/etc/profile

JAVA\_HOME=/usr/myfile/jdk1.8.0\_161

JRE\_HOME=/usr/myfile/jdk1.8.0\_161/jre

CLASS\_PATH=.:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar:$JRE\_HOME/lib

PATH=$PATH:$JAVA\_HOME/bin:$JRE\_HOME/bin

exportJAVA\_HOMEJRE\_HOMECLASS\_PATHPATH

5.刷新修改的文件使其生效 source /etc/profile

6.运行 java -version

## Linux maven 安装：

1.上传文件

2.解压

3.电脑里要有 两个插件： yum install net-tools 插件 和 wget 命令： yum install wget 插件

4.修改settings文件：

仓库位置不用修改

添加镜像即可：

位置：conf/settings.xml

<mirror>

<id>alimaven</id>

<name>aliyun maven</name>

<url>http://maven.aliyun.com/nexus/content/groups/public/</url>

<mirrorOf>central</mirrorOf>

</mirror>

5.给maven添加环境变量：

MAVEN\_HOME=/root/java/maven

path=$path:$MAVEN\_HOME/bin

6.测试运行：mvn -v

Linux Tomcat 安装：

1.上传文件

2.解压

3.启动 Tomcat服务器 ： ./要运行的文件 ./startup.sh

4.修改服务器端口：

文件位置：tomcat/conf/server.xml

修改 <Connector port="8081(端口名)" pro....

5.关闭Tomcat服务器 ：./shutdown.sh

6.Tomcat开机自启动：

<1> 修改 vi /etc/rc.d/rc.local //rc.local文件开机会自己加载里面的内容，是专门用来设置开机自启动的文件

添加自启动变量配置：

<!---- java的配置路径 ---->

export JAVA\_HOME=/root/java/jdk1.8.0\_161/bin

export JRE\_HOME=/root/java/jdk1.8.0\_161/jre

<!---- 执行Tomcat启动文件 ---->

sh /root/java/tomcat/bin/startup.sh start

<2> 给文件设置可执行的权限

chmod +x /etc/rc.d/rc.local

<3> 重启电脑，查看进程

ps -ef|grep tomcat

安装总结： 解压命令:tar–zxvf文件名

mv源文件名改后名

环境变量:

/etc/profile文件

在profile文件中设置变量时变量名=变量值

=两边没有空格

获取之前的变量值$

拼接变量用:

export命令是将变量生效

更改完profile文件后刷新:source profile】

7.安装多个Tomcat：

<1>有几个Tomcat就配置几个几组环境变量：

####FirstTomcat####

export CATALINA\_BASE=/root/java/tomcat

export CATALINA\_HOME=/root/java/tomcat

export TOMCAT\_HOME=/root/java/tomcat

####SecondTomcat####

export CATALINA\_2\_BASE=/root/java/tomcat2

export CATALINA\_2\_HOME=/root/java/tomcat2

export TOMCAT\_2\_HOME=/root/java/tomcat2

####ThirdTomcat####

export CATALINA\_3\_BASE=/root/java/tomcat3

export CATALINA\_3\_HOME=/root/java/tomcat3

export TOMCAT\_3\_HOME=/root/java/tomcat3

配置完后用 source /etc/profile 命令使其生效

<2>然后在每个Tomcat 的 bin 目录下的 catalina.sh 文件中：

#OSspecificsupport.$var\_must\_besettoeithertrueorfalse.

并在上述代码下面添加以下代码，目的是为了tomcat1启动的时候能通过该文件的里的

$CATALINA\_BASE和$CATALINA\_HOME找到/etc/profile配置文件里我们前一步配好的多个Tomcat的

具体路径，完成tomcat2的启动过程。

#myself:add

exportCATALINA\_BASE=$CATALINA\_2\_BASE

exportCATALINA\_HOME=$CATALINA\_2\_HOME

如果有多个tomcat，修改方法是一样的。

<3>启动Tomcat服务器

./startup.sh

关闭Tomcat服务器

./shutdown.sh

## Nginx 安装：

1.nginx 安装环境：

四个nginx依赖的插件

<1> yum install gcc-c++ |或| yum install -y make cmake gcc gcc-c++ （主要选后者）

//安装nginx需要先将官网下载的源码进行编译，编译依赖gcc环境

<2> yum install -y pcre pcre-devel

//PCRE(PerlCompatibleRegularExpressions)是一个Perl库，包括perl兼容的正则表达式

库。nginx的http模块使用pcre来解析正则表达式，所以需要在linux上安装pcre库。

注：pcre-devel是使用pcre开发的一个二次开发库。nginx也需要此库。

<3> yum install -y zlib zlib-devel

//zlib库提供了很多种压缩和解压缩的方式，nginx使用zlib对http包的内容进行gzip，所

以需要在linux上安装zlib库。

<4> yum install -y openssl openssl-devel

//OpenSSL是一个强大的安全套接字层密码库，囊括主要的密码算法、常用的密钥和证书

封装管理功能及SSL协议，并提供丰富的应用程序供测试或其它目的使用。

nginx不仅支持http协议，还支持https（即在ssl协议上传输http），所以需要在linux

安装openssl库。

<以上总结> ：

yum install gcc gcc-c++

yum install pcre-devel

yum install zlib zlib-devel

yum install openssl openssl-devel

也可以写成一句命令：yum install gcc-c++ pcre-devel zlib zlib-devel openssl openssl-devel 直接执行即可

【注意：安装插件命令中 yum install -y gcc-c++ 中 -y 的作用是遇到[y/d/N]选项时会自动选择 y 选项】

【注意：quit;命令 可以退出编辑字符串命令】

【注意：ln 命令：

语法格式：ln [参数][源文件或目录][目标文件或目录]

[参数]解释：必要参数：

-b 删除，覆盖以前建立的链接

-d 允许超级用户制作目录的硬链接

-f 强制执行

-i 交互模式，文件存在则提示用户是否覆盖

-n 把符号链接视为一般目录

-s 软链接(符号链接)

-v 显示详细的处理过程

给文件创建软链接，为文件创建软链接，如果源文件丢失，目标文件将失效

】

2.编译安装：

进入nginx解压文件内部：

执行：./configure && make && make install 命令，编译安装

完成后，会在/usr/local/nginx下，文件内部生成相应的可执行文件、配置、默认站点

等文件，【注意：生成的文件不是在/root/java/nginx下的】，命令也可以一个一个执行，不用&&符号连接即可

<解释>：

./configure 编译

make 加载make

make install 安装 nginx 软件

&&并且 等同于并且执行下一个命令，可以&&符号连接多个命令

3.快捷键启动关闭Nginx ：启动Nginx：先到 usr/local/nginx/sbin 目录下 执行 ./nginx -c /usr/local/nginx/conf/nginx.conf 文件 启动成功！

-c 指定启动，并执行配置文件；

启动报错：在Nginx文件夹内创建，mkdir logs 文件 ，

关闭Nginx： 快速停止：./nginx-s stop

完整停止(建议使用)：./nginx-s quit

重启Nginx：1.先停止再启动：

2.重新加载配置文件，相当于重启 ./nginx -s reload 命令

.....更多命令可在百度上查找

4.设置文件的访问权限 ：

路径和命令：chmoda+x/etc/init.d/nginx (a+x==>allusercanexecute所有用户可执行)

这样在控制台就很容易的操作nginx了：查看Nginx当前状态、启动Nginx、停止Nginx、重

启Nginx…

5.Nginx的主要配置介绍 ：

[root@zhiyou01conf]#vinginx.conf

#全局区有一个工作子进程，一般设置为CPU数\*核数

worker\_processes1;

events{

#一般是配置nginx进程与连接的特性

#如1个word能同时允许多少连接，一个子进程最大允许连接1024个

连接

worker\_connections1024;

}

#配置HTTP服务器配置段

http{

#配置虚拟主机段

server{

#定位，把特殊的路径或文件再次定位。

location{

}

}

}

6.给Nginx 集群配置多个Tomcat服务器：

配置路径 ： nginx/conf/nginx.conf 文件；

upstream mynginx{ #集群地址

server 192.168.75.128:8081 weight=1;

#单个服务地址及其权重，这个权重是默认负载均衡算法的值，越大越容易被访问

server 192.168.75.128:8082 weight=5;

server 192.168.75.128:8083 weight=10;

}

在location/{

proxy\_pass http://mynginx; //配置的多个服务器

}

7.给Nginx设置虚拟主机 ：

虚拟主机 ：一个主机代表一个网站

【注意：一个server代表一个虚拟主机】

配置：

基于域名的虚拟主机：

server{

listen80;#端口

server\_namezhiyou\_java;#域名

location/{

rootabc;#相当于ningx目录下的abc目录，可以写成绝对路径/zhiyou/java

indexa.html;#默认跳转页面

}

}

虚拟主机运行需要配置 本地的host文件：

C:\Windows\System32\drivers\etc\hosts

http请求->本机hosts->网络dns->

基于端口的虚拟主机：

server{

listen81;#端口

server\_name zhiyou\_java;#域名

location/{

rootabc;#相当于ningx目录下的abc目录，可以写成绝对路径/zhiyou/java

indexb.html;#默认跳转页面

}

}

基于IP的虚拟主机：

server{

listen80;#端口

server\_name192.168.233.129;#ip可以是任意电脑上的ip

location/{

root/usr/;#此处写的是绝对路径/usr/下c.html，需要先创建此文件

indexc.html;#默认跳转页面

}

}

8.Nginx实现动静分离 ：

静态资源由Nginx服务器托管

动态资源由Tomcat服务器托管

9.查看日志：

查看三个tomcat日志：

cat/usr/local/tomcat/logs/localhost\_access\_log.2019-09-21.txt

cat/usr/local/tomcat2/logs/localhost\_access\_log.2019-09-21.txt

cat/usr/local/tomcat3/logs/localhost\_access\_log.2019-09-21.txt

## FastDFS/文件存储器 安装：

1.安装环境：

<1>编译依赖需要gcc插件：

执行：yum install gcc-c++

<2>FastDFS依赖libevent库，需要安装：

执行：yum install libevent

<3>下载libfastcommon

libfastcommon是FastDFS官方提供的，libfastcommon包含了FastDFS运行所需要的一些基础库。

执行：wget https://github.com/happyfish100/libfastcommon/archive/V1.0.7.tar.gz 命令，自动根据网址下载tar包

wget命令连接网站失败时 在后面加 -e robots=off 即：

wget -e robots=off https://github.com/happyfish100/libfastcommon/archive/V1.0.7.tar.gz 用此方式可连接成功！

总结：安装环境：yum install gcc-c++

yum install libevent

wget -e robots=off https://github.com/happyfish100/libfastcommon/archive/V1.0.7.tar.gz

1)解压V1.0.7.tar.gz

进入libfastcommon-1.0.7

编译 ./make.sh

安装 ./make.sh install //install 在命令的后面代表 [安装]

2）libfastcommon.so安装到了/usr/lib64/libfastcommon.so，但是FastDFS主程序设置的lib目录

是/usr/local/lib，也需要这个文件，所以需要创建软链接

#ln -s /usr/lib64/libfastcommon.so /usr/local/lib/libfastcommon.so

#ln -s /usr/lib64/libfastcommon.so /usr/lib/libfastcommon.so

#ln -s /usr/lib64/libfdfsclient.so /usr/local/lib/libfdfsclient.so

#ln -s /usr/lib64/libfdfsclient.so /usr/lib/libfdfsclient.so

<4>修改hosts文件，此文件是IP文件：【hosts文件作用：能加快浏览器对域名的解析 格式：IP 空格 网址/域名】

#vim /etc/hosts

2.tracker编译安装 ：

1）将FastDFS\_v5.05.tar.gz拷贝至 /usr/local 下

解压文件

进入FastDFS内部进行编译安装

./make.sh 编译

./make.sh install 安装

## 3.FastDFS java API 上传演示：

1.导包，即，pom依赖：

<dependency>

<groupId>net.oschina.zcx7878</groupId>

<artifactId>fastdfs-client-java</artifactId>

<version>1.27.0.0</version>

</dependency>

2.写配置文件 : fastdfs-client.properties

connect\_timeout=30

network\_timeout=30

charset=UTF-8

http.tracker\_http\_port=80 //端口号

tracker\_server=192.168.87.128:22122 //IP地址

3.java 测试类 ：

packagecom.zhiyou100.test;

importjava.io.IOException;

importorg.csource.common.MyException;

importorg.csource.common.NameValuePair;

importorg.csource.fastdfs.ClientGlobal;

importorg.csource.fastdfs.StorageClient1;

importorg.csource.fastdfs.TrackerClient;

importorg.csource.fastdfs.TrackerServer;

publicclassTestFastDFS{

public static void main(String[]args)throws IOException,MyException{

//1.加载配置文件

ClientGlobal.init("E://B-TeachDemoSpace//TestFastDFS//src//main//resources//fastdfs-client.properties");

//2.创建管理端对象

TrackerClient trackerClient = new TrackerClient();

//3.通过管理端对象获取连接

TrackerServer connection = trackerClient.getConnection();

//4.创建存储端对象

StorageClient1 storageClient = new StorageClient1(connection,null);

//创建文件属性信息对象数组

//NameValuePairmeta\_list[] = new NameValuePair[]{

//

//newNameValuePair("item\_id","100010"),

//newNameValuePair("width","80"),

//newNameValuePair("height","90")

//};

NameValuePair[]meta\_list=newNameValuePair[1];

meta\_list[0]=newNameValuePair("fileName","2.jpg");

//5.上传文件

/\*

\*参数1:要上传的文件地址

\*参数2:要上传的文件类型

\*参数3:文件属性信息对象数组

\*返回值:存储在Storage中的地址

\*/

Stringpath=storageClient.upload\_file1("C://Users//User//Desktop//2.jpg","jpg",meta\_list);

System.out.println("path:"+path);

System.out.println("结束");

}

}

## 4.FastDFS和Nginx整合：

<1>上传文件到FastDFS：

1）需要修改 /usr/local/Nginx 路径下的配置文件中的关联FastDFS文件存储器的代码：作用：关联，

添加如下行，将/group1/M00映射到/home/FastDFS/fdfs\_storage/data

location /group1/M00{

alias/home/FastDFS/fdfs\_storage/data;

}

2）给上传按钮绑定点击事件：onclik="doUpload()"

function doUpload(){

varformData=newFormData($("#uploadForm")[0]);

$.ajax({

url:'${pageContext.request.contextPath}/fileupload.do',

type:'POST',

data:formData,

async:false,

cache:false,

contentType:false,

processData:false,

success:function(data){

alert(data);

$("#imgid").attr("src",data);

}

});

}

项目连接FastDFS：

1）创建配置文件：fastdfs-client.properties 文件名

connect\_timeout = 30

network\_timeout = 30

charset = UTF-8

http.tracker\_http\_port = 80 //端口号

tracker\_server = 192.168.87.128:22122 //要连接的主机的IP地址

2）创建FastDFSUtil工具类：

public class FastDFSUtil{

public static String upload(MultipartFile file)throwsException{

//获得文件名

String fileName = file.getOriginalFilename();

//获得文件后缀

String[] split = fileName.split("\\.");

String suffix = split[1];

System.out.println("FastDFSUtil:文件名:"+fileName+",后缀:"+suffix);

//1.加载配置文件

ClientGlobal.init("D:/000\_J\_A\_V\_A/A-Workspace/B-TeachDemoSpace/TestFastDFS\_Redi

s/src/main/resources/fastdfs-client.properties");

//2.创建管理端对象

TrackerClient trackerClient = new TrackerClient();

//3.通过管理端对象获取连接

TrackerServer connection = trackerClient.getConnection();

//4.创建存储端对象

StorageClient1 storageClient = new StorageClient1(connection,null);

NameValuePair[] meta\_list = new NameValuePair[1];

//需要传入文件名

meta\_list[0] = new NameValuePair("fileName",fileName);

//5.上传文件

/\*

\*参数1:要上传的文件的字节数组

\*参数2:要上传的文件类型

\*参数3:文件属性信息对象数组

\*返回值:存储在Storage中的地址

\*/

Stringpath=storageClient.upload\_file1(file.getBytes(),suffix,meta\_list);

System.out.println("FastDFSUtil-path:http://192.168.87.128/"+path);

return"http://192.168.87.128/"+path;

}

}

3）项目接收页面的请求：

@RequestMapping("/fastDFS")

@ResponseBody //让返回的数据变成字符串类型

public String fastDFSupload(MultipartFile file){

//调用FastDFSUtil方法，传入文件对象，返回一个FastDFS文件ID

String upload = FastDFSUtil.upload(file);

//返回文件服务器内的文件ID路径

return upload;

}

【注意：MultipartFile 是文件对象，包括文件的所有信息】

【FastDFS上传总结：1.获取MultipartFile（即文件对象），获取要上传的文件路径，

2.从MultipartFile中获取的文件名和文件路径，传入FastDFS中

3，FastDFS返回：存储在Storage中的地址，下次找此文件时可用此地址(即文件ID)查找】

<2>从FastDFS下载文件到本地：

1)//FastDFS工具类 ：作用：通过文件ID获取文件的字节码文件并返回

public class FastDFSUtil{

publicstaticbyte[]download(Stringfid)throwsIOException,MyException{

System.out.println("接收到的文件id"+fid);

//1.加载配置文件

ClientGlobal.init("D:/000\_J\_A\_V\_A/A-Workspace/B-TeachDemoSpace/TestFastDFS\_Redi

s/src/main/resources/fastdfs-client.properties");

//2.创建管理端对象

TrackerClienttrackerClient=newTrackerClient();

//3.通过管理端对象获取连接

TrackerServerconnection=trackerClient.getConnection();

//4.创建存储端对象

StorageClient1storageClient=newStorageClient1(connection,null);

//5.通过文件id得到文件的字节数组

byte[]bytes=storageClient.download\_file1(fid);

return bytes;

}

}

【FastDFS下载总结：从页面的Ajax中获取文件id和下载路径,用文件id找到文件并获取字节码文件，

用字节输出流对象，根据下载路劲把字节码文件输出到文件夹中】

## Linux系统MySQL：

1.Linux系统安装Mysql：

2.Mycat实现数据分类存储：和Nginx差不多实现负载均衡

例如：两台服务器，两个MYSQL数据库，一个存用户，一个存用户数据

【Mycat安装配置可百度搜索进行配置】

3.MySQL主从复制：

解释：多台数据库服务器，从主数据库服务器上复制一模一样的数据

实现： 1.环境准备：

LinuxMySQL安装： 主 从

位置 /usr/local/mysql /usr/local/mysql

IP 192.168.75.128 192.168.75.129

2.主服务器配置：

【mysql的主从复制的配置可百度搜索进行配置】

## redis 安装到Linux系统 ：

1.解压Redis文件

2.进入Redis文件夹中，执行make命令进行编译（由于底层是用C语言编写的，所以要用make命令编译过后才能安装）

3.进入到src下执行make install命令，安装Redis

4.首先为了方便管理，将Redis文件中的conf配置文件和常用命令移动到统一文件中

a)创建bin和etc文件夹

mkdir /usr/local/redis/bin

mkdir /usr/local/redis/etc

b)执行Linux文件移动命令：

mv /usr/local/redis/redis.conf /usr/local/redis/etc

cd /usr/local/redis/src

mv mkreleasehdr.sh redis-benchmark redis-check-aof redis-cli redis-server /usr/local/redis/bin

5.设置后台启动Redis

a)首先编辑conf文件，将daemonize属性改为yes（表明需要在后台运行）

cd etc/

Vim redis.conf

修改redis.conf配置文件:

## 为了redis客户端远程能够访问

1.将`bind 127.0.0.1`改为`#bind 127.0.0.1`

2.将`protected-mode yes`改为`protected-mode no`,

## 指定日志文件目录 ,暂时可以不设置

## logfile "/usr/local/redis/server-out.log"

## 默认启动时为后台启动 原来是no,

daemonize yes

b)再次启动redis服务端，并指定启动服务配置文件

redis-server /usr/local/redis/etc/redis.conf

6.服务端启动成功后，再执行Redis-cli命令，启动客户端

quit 退出客户端

7.设置Redis客户端登录密码

a)修改 redis/etc/redis.conf 文件配置

打开配置文件.在/redis/etc下的redis.conf

vim redis.conf

找到#requirepass foobared

去掉行前的注释，并修改密码为所需的密码,保存文件

requirepass 123456

b)重启服务

关闭Redis

redis-cli shutdown

关闭服务端

pkill redis-server

开启服务(指定配置文件的形式)

redis-server /usr/local/redis/etc/redis.conf

<1>进入客户端 不使用密码

redis-cli

可以进入,但是不能执行命令

<2>进入客户端 使用密码（建议使用）

redis-cli –a 123456

可以进入,并且可以执行命令

8. Redis 操作常用命令

Redis-cli 启动redis 客户端

Pkill redis-server 关闭redis服务

Redis-cli shutdown 关闭redis客户端

Netstat -tunpl|grep 6379 查看redis 默认端口号6379占用情况

## redis 的数据类型 ：

Redis中存储数据是通过key-value存储的，对于value的类型有以下几种：

字符串

Hash类型

List

Set

SortedSet（zset）

在redis中的命令语句中，命令是忽略大小写的，而key是不忽略大小写的。

### 1.string 类型：

#### <1>设置单个值

set key value 返回 OK

get key value 返回 OK

#### <2>设置多个值

mset key value [key value ...] 返回 OK

mget key [key ...] 返回 OK

#### <3>取值并赋值

getset key value 设置新值 返回 取出的值

#### <4>删除值

del key 返回(integer)1 影响行数

#### <5>数值递增

incr key 让当前的值增加，并返回递增后的值

#### <6>数值增加指定的整数

incrby key [decrement(指定每次减少多少个)]

#### <7>数值递减

decr key 让当前的值减少，并返回递增后的值

#### <8>数值减少指定的整数

decrby key [decrement(指定每次减少多少个)]

#### <9>追加值

append key value(要追加的值)

#### <10>获取字符串字符长度

strlen key 返回 字符串的长度，没有这个键就返回 0

#### <11>应用范围：定义商品编号：

设置自增主键：

incr (item:id)

商品编号、订单号采用string的递增数字特性生成。

定义商品编号key：items:id

### 2.hash类型：

#### <1>设置一个hash，在hset中设置键值 key value ，一次只能设置一个

hset (定义的hash名字) (存储键值对[key value])

#### <2>设置多个hash

hmset (定义的hash名字) (存储键值对[key value key value ....])

#### <3>当字段不存在时赋值，类似HSET，区别在于如果字段存在，该命令不执行任何操作

语法：hsetnx (定义的hash名字) (存储键值对[key value])

#### <4>取值只能取一个

hget (定义的hash名字) (hset中存储的key)

#### <5>取出多个值

hmget (定义的hash名字) (hset中存储的[key key ...])

#### <6>获取所有字段值

hgetall (定义的hash名字)

#### <7>删除一个或者多个

hdel (定义的hash名字) (hset中存储的[key key ...])

#### <8>增加数字

hincrby (定义的hash名字) (hset中存储的key) (要增加的数字)

#### <9>判断hset中字段是否存在

hexists (定义的hash名字) (hset中存储的键值对[key])

#### <10>获取hash中的所有key

hkeys (hash名字)

#### <11>获取hash中的所有value

hvals (hash名字)

#### <12>获取hash中字段的数量

hlen (hash名字)

#### <13>应用范围：定义商品信息：

商品字段

【商品id、商品名称、商品描述、商品库存、商品好评】

定义商品信息的key

商品1001的信息在Redis中的key为：[items:1001]

### 3.list类型：

#### <1>?向列表左边增加元素

lpush （List表名） [value,value ...]

例如：lpush list:1 1 2 3 // list:1 是一个整体，是一个表名

#### <2>?向列表右边增加元素

rpush （List表名） [value,value ...]

例如：rpush list:1 4 5 6

#### <3>查全部

lrange（List表名） [从哪个下标开始，到哪个下标结束]

列如： lrange list:1 0 2

#### <4>从列表左边弹出一个元素，返回被移除的元素值（弹出元素等于删除，列表中不会再有该值出现）

lpop（弹出的元素个数）

例如：lpop list:1

#### <5>从列表右边弹出一个元素，返回被移除的元素值（弹出元素等于删除，列表中不会再有该值出现）

rpop（弹出的元素个数）

例如：rpop list:1

#### <6>获取列表中元素的个数

llen （List表名）

例如：llen list:1

#### <7>删除列表中指定的值

LREM命令会删除列表中前count个值为value的元素，返回实际删除的元素个数。根据count值的不同，该命令的执行方式会有所不同：

?当count>0时， LREM会从列表左边开始删除。

?当count<0时， LREM会从列表后边开始删除。

?当count=0时， LREM删除所有值为value的元素。

语法：LREM key count value

例如:lrem （表名）（负数从右边删除，正数从左边删除，不管正数还是负数数字是几就删除几个相同的值）（要删除的值）

#### <8>?获得指定索引的元素值

语法：lindex（表名）（想要获取的值的下标）

例如： lindex l:list 2

#### <9>设置指定索引的元素值

lset（表名）（要更改的值的下标） （要更改的值）

例如： lset l:list 2 2

#### <10>只保留列表指定片段

ltrim （表名） （从哪个下标开始保留） （到哪个下标结束保留）

例如：ltrim l:list 0 2

#### <11>向列表中插入元素

linsert （表名） （前面还是后面 after/before） （找到该值，在该值的前后进行插入操作） （要插入的值）

#### <12>将元素从一个列表转移到另一个列表中

rpoplpush list newlist //从list表转移到newlist表中

#### <13>应用范围：商品评论列表 ：

思路：

在Redis中创建商品评论列表

用户发布商品评论，将评论信息转成json存储到list中。

用户在页面查询评论列表，从redis中取出json数据展示到页面。

定义商品评论列表key：

商品编号为1001的商品评论key【items: comment:1001】

127.0.0.1:6379> lpush items:comment:1001 '{"id":1,"name":"商品不错，很好！！","date":1430295077289}'

### 4.set类型：

#### <1>增加/删除元素

语法：SADD key member [member ...]

增加成功返回 增加的个数,

增加不成功,即增加重复数据时 返回0

127.0.0.1:6379> sadd set a b c

(integer) 3

127.0.0.1:6379> sadd set a

(integer) 0

#### <2>删除集合元素

语法：SREM key member [member ...]

删除成功 返回删除成功的元素个数

127.0.0.1:6379> srem set c d

(integer) 1

#### <3>获得集合中的所有元素

语法：SMEMBERS key

127.0.0.1:6379> smembers set删除集合元素

语法：SREM key member [member ...]

删除成功 返回删除成功的元素个数

127.0.0.1:6379> srem set c d

(integer) 1

1) "b"

2) "a”

#### <4>判断元素是否在集合中

语法：SISMEMBER key member

127.0.0.1:6379> sismember set a

(integer) 1

127.0.0.1:6379> sismember set h

(integer) 0

### 5.集合运算命令：

#### <1>集合的差集运算

语法：SDIFF key [key ...]

127.0.0.1:6379> sadd setA 1 2 3

(integer) 3

127.0.0.1:6379> sadd setB 2 3 4

(integer) 3

127.0.0.1:6379> sdiff setA setB

1) "1"

127.0.0.1:6379> sdiff setB setA

1) "4"

#### <2>集合的交集运算 A ∩ B

语法：SINTER key [key ...]

127.0.0.1:6379> sinter setA setB

1) "2"

2) "3"

#### <3>集合的并集运算 A ∪ B

语法：SUNION key [key ...]

127.0.0.1:6379> sunion setA setB

1) "1"

2) "2"

3) "3"

4) "4"

### 6.其他命令：

#### <1>获得集合中元素的个数

语法：SCARD key

127.0.0.1:6379> smembers setA

1) "1"

2) "2"

3) "3"

127.0.0.1:6379> scard setA

(integer) 3

#### <2>从集合中弹出一个元素(随机删除)

注意：由于集合是无序的，所有SPOP命令会从集合中随机选择一个元素弹出

语法：SPOP key

127.0.0.1:6379> spop setA

"1“

7.SortedSet类型zset

##### <1>增加元素

向有序集合中加入一个元素和该元素的分数，如果该元素已经存在则会用新的分数替换原有的分数。返回值是新加入到集合中的元素个数，不包含之前已经存在的元素。

语法：ZADD key score member [score member ...]

127.0.0.1:6379> zadd scoreboard 80 zhangsan 89 lisi 94 wangwu

(integer) 3

127.0.0.1:6379> zadd scoreboard 97 lisi

(integer) 0

##### <2>获取元素的分数

语法：ZSCORE key member

127.0.0.1:6379> zscore scoreboard lisi

"97"

##### <3>删除元素

移除有序集key中的一个或多个成员，不存在的成员将被忽略。

当key存在但不是有序集类型时，返回一个错误。

语法：ZREM key member [member ...]

127.0.0.1:6379> zrem scoreboard lisi

(integer) 1

##### <4>获得排名在某个范围的元素列表

获得排名在某个范围的元素列表

?按照元素分数从小到大的顺序返回索引从start到stop之间的所有元素（包含两端的元素）

语法：ZRANGE key start stop [WITHSCORES]

127.0.0.1:6379> zrange scoreboard 0 2

1) "zhangsan"

2) "wangwu"

3) "lisi“

按照元素分数从大到小的顺序返回索引从start到stop之间的所有元素（包含两端的元素）

语法：ZREVRANGE key start stop [WITHSCORES]

127.0.0.1:6379> zrevrange scoreboard 0 2

1) " lisi "

2) "wangwu"

3) " zhangsan “

如果需要获得元素的分数的可以在命令尾部加上WITHSCORES参数

127.0.0.1:6379> zrange scoreboard 0 1 WITHSCORES

1) "zhangsan"

2) "80"

3) "wangwu"

4) "94"

##### <5>获得指定分数范围的元素

语法：ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT offset count]

127.0.0.1:6379> ZRANGEBYSCORE scoreboard 90 97 WITHSCORES

1) "wangwu"

2) "94"

3) "lisi"

4) "97"

127.0.0.1:6379> ZRANGEBYSCORE scoreboard 70 100 limit 1 2

1) "wangwu"

2) "lisi"

##### <6>增加某个元素的分数

返回值是更改后的分数

语法：ZINCRBY key increment member

127.0.0.1:6379> ZINCRBY scoreboard 4 lisi

"101“

##### <7>获得集合中元素的数量

语法：ZCARD key

127.0.0.1:6379> ZCARD scoreboard

(integer) 3

##### <8>获得指定分数范围内的元素个数

语法：ZCOUNT key min max

127.0.0.1:6379> ZCOUNT scoreboard 80 90

(integer) 1

##### <9>按照排名范围删除元素

语法：ZREMRANGEBYRANK key start stop

127.0.0.1:6379> ZREMRANGEBYRANK scoreboard 0 1

(integer) 2

127.0.0.1:6379> ZRANGE scoreboard 0 -1

1) "lisi"

##### <10>按照分数范围删除元素

语法：ZREMRANGEBYSCORE key min max

127.0.0.1:6379> zadd scoreboard 84 zhangsan

(integer) 1

127.0.0.1:6379> ZREMRANGEBYSCORE scoreboard 80 100

(integer) 1

##### <11>获取元素的排名

?从小到大

语法：ZRANK key member

127.0.0.1:6379> ZRANK scoreboard lisi

(integer) 0

?从大到小

语法：ZREVRANK key member

127.0.0.1:6379> ZREVRANK scoreboard zhangsan

(integer) 1

##### <12>应用:商品销售排行榜

需求：根据商品销售量对商品进行排行显示

思路：定义商品销售排行榜（sorted set集合），Key为items:sellsort，分数为商品销售量。

写入商品销售量：

?商品编号1001的销量是9，商品编号1002的销量是10

192.168.101.3:7007> ZADD items:sellsort 9 1001 10 1002

?商品编号1001的销量加1

192.168.101.3:7001> ZINCRBY items:sellsort 1 1001

?商品销量前10名：

192.168.101.3:7001> ZRANGE items:sellsort 0 9 withscores

### 8.Keys命令:

设置key的生存时间

Redis在实际使用过程中更多的用作缓存，然而缓存的数据一般都是需要设置生存时间的，即：到期后数据销毁。

EXPIRE key seconds 设置key的生存时间（单位：秒）key在多少秒后会自动删除

TTL key 查看key剩余的生存时间

PERSIST key 清除生存时间

PEXPIRE key milliseconds 生存时间设置单位为：毫秒

例子：

192.168.101.3:7002> set test 1 设置test的值为1

OK

192.168.101.3:7002> get test 获取test的值

"1"

192.168.101.3:7002> EXPIRE test 5 设置test的生存时间为5秒

(integer) 1

192.168.101.3:7002> TTL test 查看test的生于生成时间还有1秒删除

(integer) 1

192.168.101.3:7002> TTL test

(integer) -2

192.168.101.3:7002> get test 获取test的值，已经删除

(nil)

#### <1>keys

返回满足给定pattern 的所有key

redis 127.0.0.1:6379> keys mylist\*

1) "mylist"

2) "mylist5"

3) "mylist6"

4) "mylist7"

5) "mylist8"

#### <2>exists

确认一个key 是否存在

示例：从结果来看，数据库中不存在HongWan 这个key，但是age 这个key 是存在的

redis 127.0.0.1:6379> exists HongWan

(integer) 0

redis 127.0.0.1:6379> exists age

(integer) 1

redis 127.0.0.1:6379>

#### <3>del

删除一个key

redis 127.0.0.1:6379> del age

(integer) 1

redis 127.0.0.1:6379> exists age

(integer) 0

#### <4>rename

重命名key

示例：age 成功的被我们改名为age\_new 了

redis 127.0.0.1:6379> keys \*

1) "age"

redis 127.0.0.1:6379> rename age age\_new

OK

redis 127.0.0.1:6379> keys \*

1) "age\_new"

redis 127.0.0.1:6379>

#### <5>type

返回值的类型

示例：这个方法可以非常简单的判断出值的类型

redis 127.0.0.1:6379> type addr

string

redis 127.0.0.1:6379> type myzset2

zset

redis 127.0.0.1:6379> type mylist

list

redis 127.0.0.1:6379>

## Jedis:作用：主要用来查询Redis缓存库 JedisAPI写法：

### 1.导入maven依赖：

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>3.0.1</version>

</dependency>

### 2.在applicationContext.xml文件中配置 Redis 的配置文件 ，Jedis

<!--============================ Redis的javaAPI ：Jedis =============================-->

<!--连接池配置-->

<bean id="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig">

<!--最大连接数-->

<property name="maxTotal" value="30"/>

<!--最大空闲连接数-->

<property name="maxIdle" value="10"/>

<!--每次释放连接的最大数目-->

<property name="numTestsPerEvictionRun" value="1024"/>

<!--释放连接的扫描间隔（毫秒）-->

<property name="timeBetweenEvictionRunsMillis" value="30000"/>

<!--连接最小空闲时间-->

<property name="minEvictableIdleTimeMillis" value="1800000"/>

<!--连接空闲多久后释放,当空闲时间>该值且空闲连接>最大空闲连接数时直接释放-->

<property name="softMinEvictableIdleTimeMillis" value="10000"/>

<!--获取连接时的最大等待毫秒数,小于零:阻塞不确定的时间,默认-1-->

<property name="maxWaitMillis" value="1500"/>

<!--在获取连接的时候检查有效性,默认false-->

<property name="testOnBorrow" value="false"/>

<!--在空闲时检查有效性,默认false-->

<property name="testWhileIdle" value="true"/>

<!--连接耗尽时是否阻塞,false报异常,ture阻塞直到超时,默认true-->

<property name="blockWhenExhausted" value="false"/>

</bean>

<!--redis单机通过连接池-->

<bean id="jedisPool" class="redis.clients.jedis.JedisPool"

destroy-method="close">

<constructor-arg name="poolConfig" ref="jedisPoolConfig"/>

<constructor-arg name="host" value="192.168.75.128"/>

<constructor-arg name="port" value="6379"/>

</bean>

</beans>

### 3.java代码写法：

<1>使用自动封装获取JedisPool：

@Autowired

JedisPool jedisPool;

<2>获取Jedis连接Redis对象：

Jedis jedis = jedisPool.getResource();//得到一个连接对象

<3>利用连接对象中的方法操作Redis中的数据，基本和Redis中的命令一样：

例如：String s = jedis.get("k" + id);//根据页面传进来的key查询值

【未完成】

## Elasticsearch /数据库搜索引擎 安装：

1.解压包

2.目录：

bin/ 主要的执行命令 , 比如启动

config/ es的配置文件目录

lib/ 依赖的第三方库的目录

modules/ 模块目录

plugins/ 插件目录

3.创建用户

es 不允许root用户登录,所以创建es的用户,并授权

// 创建用户组

[root@zhiyou elasticsearch-6.3.0]# groupadd es

// 创建用户

[root@zhiyou elasticsearch-6.3.0]# useradd es -g es

4.修改es配置文件elasticsearch.yml

路径：vim /root/java/elasticsearch/config/elasticsearch.yml

在文件中添加 ：network.bind\_host: 0.0.0.0

5.启动

切换为es用户 , 进入bin目录执行命令启动es

## Git仓库：

【未完成】

## SpringBoot：

spring-boot-starter-parent这是一个依赖管理器的pom文件。它的作用就是管理boot需要的所有依赖，从而统一各种jar的版本号，避免了版本不一致而出现的问题。所以，引入其他的依赖就可以省略版本号。当然也可以加上指定的版本号，从而取代默认的。

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.0.RELEASE</version>

</parent>

　　spring-boot-starter-web会自动嵌入tomcat容器。同时，springboot也会根据classpath中的dependency来自动配置。比如：spring-boot-starter-web会自动装配tomcat容器；并且会自动从application.properties中读取web应用的配置，如：server.port；如果application.properties没有配置相关的参数，则采用默认的配置信息，如：8080。

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

　spring-boot-starter-data-jpa数据库连接的依赖：可以使用数据持久层框架 Spring Data JPA 操作数据源

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

　　fastjson依赖库：

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.2.15</version>

</dependency>

　　Mysql依赖：

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

## java配置类

## 1.配置文件 用java类的方式配置：

类上要有@Configuration注解，方法上要有@Bean注解，然后在方法中创建对象并返回即可

调用时，创建new AnnotationConfigApplicationContext()对象调用,传入配置类的.class文件

获取对象中Bean方法传入（配置类中创建对象的方法的名字，传入创建出的对象的.class文件）

//配置类的方法说明：

@Configuration

public class BeanConfig{

@Bean

public User getUser(){

return new User;

}

}

获取配置类中的创建的对象:

AnnotationConfigApplicationContext app = new AnnotationConfigApplicationContext(BeanConfig.class);

User user = (User)app.Bean("getBean",User.class);//getBean相当于<bean>标签内部的id值，也就是配置类中的方法名

system.out.println("得到的对象："+user);

## 2.SpringBoot MVC

java写法：

@RestController

public class Login{

@GetMapping("/Login")

@RequestBody

public String name(@Parm name){

return "Hollo SpringBoot";

}

}

【//@RestController 该注解加在类上，那么该类下的所有方法返回值都是json格式,

就不用每个方法上都使用@RequestBody注解了】

【@Parm("id")】

【//@ResponseBody 可以放在方法上，解释：返回的值会被解析为json字符串】

【@RequestBody 可以放在接收的参数上，解释：接收的数据会被解析为json数据,

@RequestBody接受对象只能只能接受POST或着PUT请求的对象，GET请求在url上的复杂对象并不能接收。】

【注意！：@ResponseBody ：响应解析

@RequestBody ：请求解析 】

【//@PathVariable 取出请求的路径中的值 是写在请求参数上的注解，路径中必须要以${id}的形式添加值】

//@DeleteMapping("/Login") 只接收delete方法的请求

//@GetMapping("/Login") 只接收get方法的请求

//@PutMapping("/Login") 只接收put方法的请求

//@PostMapping("/Login") 只接收post方法的请求

3.返回的json对象：

创建json对象，创建的对象里面要有：

public class Json{

private String code;//状态码

private String msg;//信息

private Object data;//要返回的数据

}

对象的属性等于值，即 key = value

对象的属性名 = 传入对象的值

json字符串：{对象的属性名 = 传入对象的值，对象的属性名 = 传入对象的值}

4.SpringBoot 的配置文件

分为两种： bootstrap.properties

application.properties

大多数情况用：application.properties文件进行配置

bootstrap.properties文件比application.properties文件优先加载，

且application.properties文件配置不能覆盖bootstrap.properties文件的配置

如果两个配置相同，以bootstrap.properties配置文件为主

5.application 文件的后缀有多种，且写法也不一样：

支持yaml yml properties 后缀格式的文件

yaml和yml文件一样

yml:以空格定层级关系

properti：直接写 点 定层级关系

6.配置文件的位置一般情况下在resources下

但是也可以有其他位置,

1.file:./config/?当前项目文件夹下的config文件夹中写配置文件

2.file:./?当前项目文件夹下写配置文件

3.classpath:/config/?当前项目类路径(resource)下的config文件夹中写

配置文件

4.classpath:/?当前项目类路径(resource)下写配置文件

以上四个位置都是SpringBoot默认加载的配置文件的位置.且从上至下加载

优先级依次降低.既

先加载1最后加载4,如果配置文件中有重复配置的东西.优先级高的会覆盖

优先级低的.

一般都是第四种,不建议使用其他的.

7.SpringBoot请求和重定向和SpringMVC一样

8.SpingBoot接收数据:

在参数上添加@RequestBody注解可以接收json数据

json数据：

<1>接收基本数据类型：

<2>接收对象：

<3>接收数组：

<4>接收Map：

<5>

## 9.SpringBoot拦截器：

<1>创建自定义拦截器类 实现接口：HandlerInterceptor{

@Component//该注解使类被Spring容器托管

publicclassMyInterceptorimplementsHandlerInterceptor{

//重新方法:拦截请求,获得登录信息

@Override

public boolean preHandle(HttpServletRequest request,HttpServletResponse response,Object handler)throwsException{

System.out.println("MyInterceptor-url="+request.getRequestURL());

return true;//放行的意思

}

}

<2>Spring配置文件，以前用Spring.xml文件进行配置，现在用java类的方式进行配置

@Configuration//该注解说明该类是一个Spring的配置类,相当于是之前的配置文件

public class MyInterceptorConfig implements WebMvcConfigurer{

//注意实现了接口

//重新添加拦截路径的方法

@Override

public void addInterceptors(Interceptor Registryregistry){

//将自己的拦截器注册进运行环境中

Interceptor Registrationregistration = registry.addInterceptor(new MyInterceptor());

///\*匹配单层路径/a或者/b可以拦截到,/a/b/c这样的多层路径不会拦截

///\*\*匹配单层路径/a或者/b多层路径/a/b/c都可以

//支持后缀匹配.注意不是\*.do而是/\*.do

//支持路径中的模糊匹配/user/\*\*

registration.addPathPatterns("/user/\*\*");

//添加一些不拦截路径

registration.excludePathPatterns("/login","/logout","/html/t1.html");

}

}

## 10.整合原生jdbc,SpringBoot会自动配置一个数据库连接池：

<1>导入依赖

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<!-- 这是阿里巴巴的Driud数据库连接池 【可有可无】-->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>1.1.10</version>

</dependency>

<2>在application配置文件中配置：

spring.datasource.username=root

spring.datasource.password=123456

#mysql驱动8版本开始,需要在路径在加入时区设置

spring.datasource.url=jdbc:mysql://localhost:3306/day27?serverTimezone=GMT

#配置了阿里的数据库连接池，有没有无所谓，没有的话SpringBoot会自动配置一个数据库连接池

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

<3>使用jdbc查询

//注入数据源

@Autowired

DataSourcedataSource;

@GetMapping("/get")

public Result getAll()throwsSQLException{

//注册驱动,(用户名,密码,数据库地址...)获得连接,

//以上操作已经在配置文件中写好,直接用

Connectionconnection=dataSource.getConnection();

Statementstatement=connection.createStatement();

ResultSetresultSet=statement.executeQuery("select\*fromusers");

//遍历,封装...

return new Result();//返回的Result对象是json数据类

}

【SpringBoot默认使用HikariDataSource连接池,我们也可以自己指定连接池】

## 11.整合MyBatis：配置MyBatis注解版：

<1>导入依赖：

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot-starter</artifactId>

<version>2.1.1</version>

</dependency>

<2>环境

需要配置jdbc的数据源已经数据库连接池.在上节课配置过,无需再配

<3>创建Mapper类

(1)在类上要加注解@Mapper 定义这个类为Mapper类

(2)在方法上添加@Select("") 注解里面写sql语句

(3)补充：在SpringBoot内部定义的Tomcat主方法上添加@MapperScan("要扫描的Mapper类的路径")

注意：主方法在所有包外面不用添加此注解，主方法会自动扫描主方法外的所有资源

(4)Mapper方法上，多个参数一定要使用@Param("参数名") 注解

java写法：

packagecom.zhiyou100.springboot02.mapper;

importcom.zhiyou100.springboot02.model.Stu;

importorg.apache.ibatis.annotations.\*;

importjava.util.List;

@Mapper//该注解说明该类就行相当于一个xxxMapper.xml文件

publicinterfaceStuMapper{

//查询全部学生

@Select("select\*fromstu")

List<Stu>findAllStu();

//根据id查一个学生

@Select("select\*fromstuwheresid=#{id}")

StufindStuBuId(intid);

//模糊查询-

//因为#{}取值出会自带引号,${}取值就是去除原值,当参数是基本数据类型,其中必须是value

@Select("select\*fromstuwheresnamelike'%${value}%'")

List<Stu>findByKeyword(Stringname);

//多条件查询年龄和性别

//Mapper接口方法中多个参数,想要注入到SQL语句中,需要使用注解@Param

@Select("select\*fromstuwhereage=#{a}andgender=#{sex}")

List<Stu>findByAgeandSex(@Param("a")intage,@Param("sex")Stringsex);

//增

@Insert("insertintostu(sid,sname)values(#{sid},#{sname})")

voidinsetStu(Stustu);

//删

@Delete("deletefromstuwheresid=#{id}")

voiddeleteById(intid);

//改

@Update("updatestusetsname=#{sname}wheresid=#{sid}")

voidupdateById(Stustu);

}

## 12.动态sql注解版：

注解实现动态sql

MyBatis提供了@XxxProvider Xxx指的是Select,Insert,Update,Delete 四种

(1).创建自定义动态sql类

创建的类中的方法直接返回一个sql字符串即可，类中方法上的参数是Mapper类中方法上的参数

(2)在Mapper类的方法上使用：@SelectProvider(自定义类的.class文件,class文件中的方法名) 注解

13.注解版–列的别名问题–结果封装

@Results注解帮助完成:和<requestMap>标签具有同样的性质

@Results({

@Result(id=true,column="虚拟表的id名",property="类的属性名"),

@Result(column="虚拟表的列名",property="类的属性名"),

@Result(column="虚拟表的列名",property="类的属性名"),

@Result(column="虚拟表的列名",property="类的属性名"),

@Result(column="虚拟表的列名",property="类的属性名"),

})

### 14.注解版一对一：

@Results注解帮助完成:和<requestMap>标签具有同样的性质,【并且也要在返回的类里面定义另一个类的属性】

@Results({

@Result(id=true,column="虚拟表的id名",property="类的属性名"),

@Result(column="虚拟表的列名",property="类的属性名"),

@Result(column="虚拟表的列名",property="类的属性名"),

@Result(column="虚拟表的列名",property="类的属性名"),

@Result(column="虚拟表的列名",property="类的属性名",one=@one(select="另一个查询数据库的sql方法路径",fetchType=fetchType.EAGER)),

})

### 15.注解版一对多：

@Results({

@Result(id=true,column="虚拟表的id名",property="类的属性名"),

@Result(column="虚拟表的列名",property="类的属性名"),

@Result(column="虚拟表的列名",property="类的属性名"),

@Result(column="虚拟表的列名",property="类的属性名"),

@Result(column="虚拟表的列名",property="类的属性名",one=@one(select="另一个查询数据库的sql方法路径",fetchType=fetchType.EAGER)),

})

## 16.Mybatis配置文件版：

<1>添加依赖：

<!--数据库驱动包-->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<!--alibaba数据库连接池-->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>1.1.10</version>

</dependency>

<!--SpringBoot整合Mybatis-->

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot-starter</artifactId>

<version>2.0.0</version>

</dependency>

<2>写mybatis-config.xml全局配置文件:

位置：放在/resources/mybatis/

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<!-- 上面的dtd是当前这个xml文件夹的约束，类似于Spring的约束 -->

<configuration>

<!-- 定义MyBatis的驼峰原则 -->

<settings>

<setting name = "mapUnderscoreToCamelCase" value = "true"/>

</settings>

<typeAliases>

<package name="comzhiyou100.Model.classRoom"></package>

</typeAliases>

</configuration>

<3>写Mapper的接口：

【和ssm的mybatis一样】

<4>Mapper接口的xml映射文件：

【和ssm的myBatis一样】

<5>Application.properties中添加mybatis的配置：

#MyBatis的全局配置文件路径

Mybatis.config-location=classpath:MyBatis的全局配置文件路径

#MyBatis的映射文件

MyBatis.mapper.-locations=classpath:mybatis/mapper/\*.xml

【路径中星号的作用是在当前路径下匹配多个映射文件属于模糊匹配的一种】

17.SpringBoot的事务：

SpringBoot以及,自动朋配置了事务.只需要在业务层的类上使用@Transactional注解即可

## 18.MyBatis日志框架：

SpringBoot默认使用logback作为日志框架.

SpringBoot使用slf4j 当做接口,使用Logback作为日志实现.

可以在application.properties文件中修改关于日志框架的设置

################ log ################

#修改log的最基础级别

logging.level.root=info

#设置log文件的输出位置，不指定绝对路径，默认输出到当前项目的文件夹下

logging.file=myapp.log

#设置日志输出到控制台的模板格式

#logging.pattern.console=[%-5p] %d{yyyy-MM-dd HH:mm:ss ,SSS} method:%l%n%m%n

#设置日志输出到控制台的模板格式

#logging.pattern.file=[%-5p] %d{yyyy-MM-dd HH:mm:ss ,SSS} method:%l%n%m%n

SpringBoot可以使用别的日志框架,不使用logback. 但是不建议使用别的.

Logback日志使用效率比其他几个日志框架更快.

## 19.SpringBoot整合Thymeleaf：

模板引擎:Thymeleaf , 就是用于渲染视图.即使用Html展现页面效果以及数据.

以前使用的jsp,但是SpringBoot不建议使用jsp,而是使用Thymeleaf.

SpringBoot支持其他的模板引擎 : Freemarker,Velocity,Groovy…

<1>添加依赖：

<!--thymeleaf-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

<2>测试:

Controller演示跳转页面,此时页面不再去/resources/static/下找. Why?

因为我们导入了Thymeleaf的依赖,SpringBoot就会自动配置静态资源使用Thymeleaf进行渲染.Thymeleaf的自动配置中默认规定,静态页面(html)应该放在/resources/template/下

且,在Controller中返回页面时,会自动拼接后缀(.html)

<3>Thymeleaf的语法格式：

整体的思想: 写html代码,然后使用Thymeleaf的语法,展现后台发送到前台的数据.

也能够做到与jsp类似的功能:

获取后端存入域(model,request,session)中的值

将值设置到html标签中或者是属性中

判断

迭代

格式化

工具类(日期,日历,字符串,数组,集合等常用工具方法)

语法格式:

在html页面标签内写 th:xxx=”表达式”

例如:获得后端存入的值

<input type=” text” th:value=”${key}”>

表达式多种:

${...} 变量表达式，Variable Expressions

\*{...} 选择变量表达式，Selection Variable Expressions

@{...} 链接表达式，Link URL Expressions

#{...} 消息表达式，Message Expressions

~{...} 代码块表达式，Fragment Expressions

Thymeleaf常用的属性

th:text 给开闭标签中设置文本

th:utext 给给你标签中设置文本,如果文本是html标签,会解析成标签

th:value 给标签中的value属性赋值

th:src 给src属性赋值

th:href …

th:attr 给指定属性赋值

th:if 判断是否 --> 是否展现被包裹的标签

th:each 遍历 th:each="good:${goods}"

Simple expressions:

(简单表达式) Variable Expressions: ${...}

Selection Variable Expressions: \*{...}

Message Expressions: #{...}

Link URL Expressions: @{...}

Fragment Expressions: ~{...}

Literals

(字面量) Text literals: 'one text' , 'Another one!' ,…

Number literals: 0 , 34 , 3.0 , 12.3 ,…

Boolean literals: true , false

Null literal: null

Literal tokens: one , sometext , main ,…

Text operations:

(文本操作符) String concatenation: +

Literal substitutions: |The name is ${name}|

Arithmetic operations:

(算术操作符) Binary operators: + , - , \* , / , %

Minus sign (unary operator): -

Boolean operations:

(布尔操作符) Binary operators: and , or

Boolean negation (unary operator): ! , not

Comparisons and equality:

(比较和判断是否相等) Comparators: > , < , >= , <= ( gt , lt , ge , le )

Equality operators: == , != ( eq , ne )

Conditional operators:

(条件操作符) If-then: (if) ? (then)

If-then-else: (if) ? (then) : (else)

Default: (value) ?: (defaultvalue)

Special tokens:(特殊操作符) No-Operation: \_ (当不想做操作时加 \_ )

Html页面想要提示Thymeleaf语法,需要在<html>标签中添加名称空间.

## thymeleaf使用方法：

<4>在<html>标签中添加 xmlns:th="http://www.thymeleaf.org 命名空间

(1)Thymeleaf操作html页面属性：

Thymeleaf常用的属性

th:text 给开闭标签中设置文本

th:utext 给给你标签中设置文本,如果文本是html标签,会解析成标签

th:value 给标签中的value属性赋值

th:src 给src属性赋值

th:href …

th:attr 给指定属性赋值

th:if 判断是否 --> 是否展现被包裹的标签

th:each 遍历 th:each="good:${goods}"

演示：

<!DOCTYPE html>

<html lang="en" xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="UTF-8">

<title>Title</title>

</head>

<body>

<p th:text="${textKey}"></p>

<p th:utext="${utextKey}"></p>

<input type="text" th:value="${valueKey}">

<img th:src="${srcKey}">

<a th:href="${hrefKey}">点我</a>

<p th:if="${ifKey == 'false'}">

看到这句话,说明判断成功

</p>

<ul>

<li th:each="val:${listKey}" th:text="${val}"></li>

</ul>

</body>

</html>

(2)Thymeleaf变量表达式(${})演示:web应用中的内置对象

【内置对象一般在表达式上使用，例如：${} 】

#代表此对象是内置对象

格式：#内置对象名 然后打点调用内置对象中的方法

#request.getAttribute()

1> Thymeleaf常见的内置对象:

#ctx

#vars

#locale

#request

#response

#session

#servletContext

2> thymeleaf其他常见的内置对象：

1) #strings：字符串格式化方法，常用的Java方法它都有。

比如：equals，equalsIgnoreCase，length，trim，toUpperCase，toLowerCase，indexOf，substring，replace，startsWith，endsWith，contains，containsIgnoreCase等

2) #numbers：数值格式化方法，常用的方法有：formatDecimal等

3) #bools：布尔方法，常用的方法有：isTrue，isFalse等

4) #arrays：数组方法，常用的方法有：toArray，length，isEmpty，contains，containsAll等

5) #lists，sets：集合方法，常用的方法有：toList，size，isEmpty，contains，containsAll，sort等

6) #maps：对象方法，常用的方法有：size，isEmpty，containsKey，containsValue等

7) #dates：日期方法，常用的方法有：format，year，month，hour，createNow等

3> ${}变量表达式 的作用就是取出 thymeleaf 中的内置对象并使用内置对象中的方法

(3)@{}连接表达式：

解释：相当于是jsp中jstl中的<c:url> . 可以给将路径自动拼接上项目名.

1.

## SwaggerApi:接口文档:

## Docker:镜像和容器:

docker常用命令:

systemctl start docker //启动docker

systemctl restart docker //重启docker

docker search 某个XXX镜像名字 # 搜索

docker pull 某个XXX镜像名字 #下载某个镜像

docker rmi 某个XXX镜像名字/ID # 删除某个镜像,默认删除latest

docker images # 列出本地主机上的镜像

docker run --name mysql -e MYSQL\_ROOT\_PASSWORD=123456 -p 3306:3306 -d mysql

? –name：给新创建的容器命名，此处命名为mysql

? -e：配置信息，此处配置mysql的root用户的登陆密码

? -p：端口映射，此处映射主机3306端口到容器mysql的3306端口 ；前面的3306是映射后的，后面的是mysql默认的

? -d：容器在后台运行, 成功启动容器后输出容器的完整ID

? 最后一个mysql指的是mysql镜像名字

docker ps //查看正在运行的容器

docker exec -it 容器名 /bin/bash //进入容器

exit //退出容器

docker stop 容器名或容器id //停止容器方式一

docker kill 容器名或容器id //停止容器方式二

docker start 容器ID或容器名 //启动容器

docker rm 容器名 //删除容器

安装docker:

1.需要gcc环境 :

yum -y install gcc

yum -y install gcc-c++

2.卸载旧的docker :

$ sudo yum remove docker \

docker-client \

docker-client-latest \

docker-common \

docker-latest \

docker-latest-logrotate \

docker-logrotate \

docker-selinux \

docker-engine-selinux \

docker-engine

3.安装必要的软件包 :

sudo yum install -y yum-utils \

device-mapper-persistent-data \

lvm2

4.设置stable镜像仓库 :

设置stable镜像仓库(注意：我们这里使用国内的镜像地址【因为Docker 官网给出的地址在国外，太慢了！】)

yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo

5.更新yum软件包索引 :

yum makecache fast

6.安装Docke ce :

yum -y install docker-ce

7.启动docker :

systemctl start docker 启动docker服务

systemctl stop docker 关闭docker服务

service docker start 启动

service docker stop 停止

service docker restart 重启

启动命令解释:

第一个代表的centos7.x的命令

第二个代表的centos6.x的命令

docker官网下载

docker启动 :

systemctl start docker

service docker start

重启docker服务 :

systemctl restart docker

sudo service docker restart

关闭docker :

systemctl stop docker

service docker stop

查看是否启动成功 :

docker ps -a

8.运行docker自带的hello-world程序:

docker run hello-world

[ 安装完成 ! ]

9.Docker [常用命令] :

docker version

docker info

docker --help

docker images # 列出本地主机上的镜像

参数-a :列出本地所有的镜像（含中间映像层） docker images –a

-q :只显示镜像ID。

--no-trunc :显示完整的镜像信息

--digests :显示镜像的摘要信息

docker search 某个XXX镜像名字 # 搜索

docker pull 某个XXX镜像名字 #下载某个镜像

docker rmi 某个XXX镜像名字/ID # 删除某个镜像,默认删除latest

docker rmi tomcat:v1.0 # 删除tag为1.0的tomcat镜像

docker rmi -f tomcat:v1.0 # 强制删除tag为1.0的tomcat镜像

docker rmi -f tomcat:v1.0 tomcat:v2.0 #强制删tag为1.0,2.0的两个tomcat镜像 空格隔开

docker run [options] imagesName # 新建并启动容器

--name="容器新名字": 为容器指定一个名称；

-d: 后台运行容器，并返回容器ID，也即启动守护式容器；

-i：以交互模式运行容器，通常与 -t 同时使用；

-t：为容器重新分配一个伪输入终端，通常与 -i 同时使用；

-P: 随机端口映射；

-p: 指定端口映射，有以下四种格式

-e: 配置信息

ip:hostPort:containerPort

ip::containerPort...

docker ps [options] #正在运行哪些docker容器

-a :列出当前所有正在运行的容器+历史上运行过的

-l :显示最近创建的容器。

-n：显示最近n个创建的容器。

-q :静默模式，只显示容器编号。

--no-trunc :不截断输出。

docker rm `docker ps -a -q` # 批量删除全部容器,符号是~处的`

exit # 退出容器

ctrl+P+Q # 容器退出不停止

docker start 容器ID或者容器名 # 重启容器

docker restart 容器ID或者容器名 # 重启容器

docker stop 容器ID或者容器名 # 停止容器

docker kill 容器ID或者容器名 # 强制停止容器

docker rm 容器ID # 删除已停止的容器

docker logs -f -t --tail 容器ID # 查看容器日志

\* -t 是加入时间戳

\* -f 跟随最新的日志打印

\* --tail 数字 显示最后多少条

docker top 容器ID # 查看容器内运行的进程

docker exec -it 容器ID /bin/bash # 进入正在运行的容器并以命令行交互

docker attach 容器ID # 进入正在运行的容器并以命令行交互

上述两个区别:

attach 直接进入容器启动命令的终端，不会启动新的进程

exec 是在容器中打开新的终端，并且可以启动新的进程

docker cp 容器ID:容器内路径 目的主机路径 # 从容器内拷贝文件到主机上

docker inspect 镜像名 # 描述镜像

10.下载镜像:

docker search 某个XXX镜像名字 # 搜索

docker pull 某个XXX镜像名字 #下载某个镜像

docker pull 某个XXX镜像名字 :5.7 #下载某个镜像,冒号后面是要下载的版本号,不跟版本号默认下载最新的版本 latest

11.运行镜像生产容器:

docker run --name mysql -e MYSQL\_ROOT\_PASSWORD=123456 -p 3306:3306 -d mysql

? –name：给新创建的容器命名，此处命名为mysql

? -e：配置信息，此处配置mysql的root用户的登陆密码

? -p：端口映射，此处映射主机3306端口到容器mysql的3306端口 ；前面的3306是映射后的，后面的是mysql默认的

? -d：容器在后台运行, 成功启动容器后输出容器的完整ID

? 最后一个mysql指的是mysql镜像名字

12.进入容器 :

docker exec -it 容器名 /bin/bash

mysql -uroot -p123456 //登录MySQL容器

13.退出容器:

exit

14.停止容器 :

停止容器有2种方式：

1、docker stop 容器名或容器id

2、docker kill 容器名或容器id

15.启动容器 :

docker start 容器ID或容器名

16.删除容器 :

docker rm 容器名

17.镜像commit : 提交容器生成新的镜像 :

是什么 ?

docker commit提交容器副本使之成为一个新的镜像

为什么 ?

比如:我们新从仓库pull拉下来的mysql,我们做过修改之后(添加了数据),重新将这一个新的mysql变为一个镜像

怎么做?

命令:

docker commit -m=“提交的描述信息” -a=“作者” 容器ID 要创建的目标镜像名:[标签名]

18.[远程连接mysql数据库,要修改的命令] :

1,容器中登录mysql,查看mysql的版本

mysql> status;

--------------

mysql Ver 8.0.11 for Linux on x86\_64 (MySQL Community Server - GPL)

2,进行授权远程连接(注意mysql 8.0跟之前的授权方式不同)授权

GRANT ALL ON \*.\* TO 'root'@'%';

3.刷新权限 : flush privileges;

此时,还不能远程访问,因为Navicat只支持旧版本的加密,需要更改mysql的加密规则

4.更改加密规则 :

ALTER USER 'root'@'localhost' IDENTIFIED BY 'password' PASSWORD EXPIRE NEVER;

5.更新root用户密码 :

ALTER USER 'root'@'%' IDENTIFIED WITH mysql\_native\_password BY '123456';

6.刷新权限 :

flush privileges;

7.重新启动MySQL容器即可 !

## mongodb : docker安装方法:

docker pull mongo:3.6

docker run -di --name=mongo -p 27017:27017 mongo:3.6

## redis : docker安装方法:

docker pull redis:5.0.5

docker run -p 6379:6379 -v /c/docker.image.data/redis/data:/data -d redis:5.0.5 redis-server --appendonly yes

进入redis 控制台

docker exec -it 2498a3301c34 redis-cli

进入后可执行redis的数据命令

## FastDFS : docker安装方法:

docker pull morunchang/fastdfs

docker run -d --name tracker --net=host morunchang/fastdfs sh tracker.sh

1.参考网站: https://blog.csdn.net/wo541075754/article/details/89607318

docker pull season/fastdfs

2.目录配置:

在宿主机中新建目录用,于存放fastdfs配置文件和数据。名称根据自己需求，如下只是样例，fastdfs在一台服务器支持多个store\_path，每个store\_path指向一个存储路径。

mkdir /usr/local/fastdfs/etc/

mkdir /usr/local/fastdfs/data/storage\_data

mkdir /usr/local/fastdfs/data/store\_path

mkdir /usr/local/fastdfs/data/tracker\_data

创建的配置的文件的作用:

etc：配置文件地址

storage\_data：存储数据地址

tracker\_data：存储数据地址

store\_path：扩容

SpringCloud:

Eureka:管理微服务

Ribbon:服务器相互之间访问

Feign:服务器相互访问

Hystrix:熔断

Zuul:网关

SpringCloudConfig:管理微服务的配置文件

## SpringBoot-jpa使用：

1.在application配置文件中写jpa的配置：

jpa: # SpringDataJPA的 配置

database: mysql

show-sql: true

此语法格式只在yml文件中适用；

2.在数据库类上写注解：

@Entity // 声明这是一个JPA的实体类 与 数据表对应

@Id //声明这是数据表中的id

上方这两个注解必须写；

需要时写：

类名要和数据库一样，属性名也要和数据库一样

不一样时可以写注解指定：

@table //指定表名，写在类上

@Column //指定数据库表的列名，写在除id属性外的属性上

3自定义接口类：

接口继承JpaRepository<要操作的model类也是数据库中的表类,id的类型>和JpaSpecificationExecutor<要操作的model类也是数据库中的表类>两个类

接口内部不用写任何东西

例如： public interface BaesRepository extends JpaRepository<TbLabel,String>,JpaSpecificationExecutor<TbLabel> {

}

4.操作自定义接口类：可直接在业务层操作自定义类，不用实现，也不用写方法和sql语句

例如： @Autowired

BaesRepository baesRepository;

查询： baesRepository.find\*\*\*();

增加： baesRepository.save(传入要增加的数据)

更新： baesRepository.save(传入要更新的数据)；//传入的id值要提前定义好，jpa会自动根据id查询要修改的是哪条数据

删除： baesRepository.deleteById(传入id根据id进行删除)

全局异常处理：

为了使我们的代码更容易维护，我们创建一个类集中处理异常在com.zhiyou100.controller包下创建公共异常处理类BaseExceptionHandler

/\*\*

\* @Author QiuShiju

\* @Desc 统一异常处理

\*/

@ControllerAdvice

public class BaseExceptionHandler {

@ExceptionHandler(Exception.class)

@ResponseBody

public ResultObject error(Exception e) {

e.printStackTrace();

return new ResultObject(false, StatusCode.ERROR,e.getMessage());

}

}

@ControllerAdvice是springmvc controller增强器。可以实现一些在原来的Controller中实现不了或者是重复的功能抽取。

https://www.cnblogs.com/sjqq/p/10103198.html

@ExceptionHandler是spring里面的一个注解，非常方便去掉try...catch这类冗余难看的代码，有利于代码的整洁和优美。还能统一处理异常，是个非常好的技巧。

配合@ControllerAdvice 或者是@RestControllerAdvice 可以实现全局异常拦截。

## 全局异常处理:

创建全局异常处理类:

@ControllerAdvice //在spring容器里面定义这是一个全局异常处理类

public class BaseExceptionHandler {

@ExceptionHandler(Exception.class) //可以省去try/cath代码

@ResponseBody //返回的数据解析成json字符串

public ReslutObject BaseExceptionHandler(){

return new ReslutObject(false,StatusCode.ERROR,"请求失败！");

}

}