1. 基础语法
2. 数据类型(内置数据类型、引用数据类型)

整数类型： byte：-128~127 二进制位数8

Short：16

Int：32

Long：默认0L 64

浮点类型： float：默认0.0f 32

Double 64

字符类型：char 16

布尔类型：boolean 默认false

引用数据类型：String、Long、Integer等

对象、数组都是引用数据类型。

所有引用类型的默认值都是null。

低 ------------------------------------> 高

byte,short,char—> int —> long—> float —> double

Char和int转换：A~Z：65-90；a~z:97-122

1. 修饰符
2. 访问修饰符

default (即缺省，什么也不写）: 在同一包内可见，不使用任何修饰符。使用对象：类、接口、变量、方法。

private : 在同一类内可见。使用对象：变量、方法。 注意：不能修饰类（外部类）

public : 对所有类可见。使用对象：类、接口、变量、方法

protected : 对同一包内的类和所有子类可见。使用对象：变量、方法。 注意：不能修饰类（外部类）。

1. 非访问修饰符

static 修饰符，用来修饰类方法和类变量。静态方法不能使用类的非静态变量。

final 修饰符，用来修饰类、方法和变量，final 修饰的类不能够被继承，修饰的方法不能被继承类重新定义，修饰的变量为常量

abstract 修饰符，用来创建抽象类和抽象方法。

抽象方法的具体实现由子类提供。

抽象方法不能被声明成 final 和 static。

任何继承抽象类的子类必须实现父类的所有抽象方法，除非该子类也是抽象类。

如果一个类包含若干个抽象方法，那么该类必须声明为抽象类。抽象类可以不包含抽 象方法。

synchronized 修饰符，主要用于线程的编程，同一时间只能被一个线程访问。

1. 运算符
2. 算数运算符

+、-、\*、/、%、++、--

1. 关系运算符

==、！=、>、<、>=、<=

1. 逻辑运算符

&&、||、！

1. 赋值运算符

=、+=、-=

1. 条件运算符（三元运算符）int c=a>b?a:b
2. 条件及循环
3. 条件

if(布尔表达式) { //如果布尔表达式为true将执行的语句 }else{表达式为false将执行的语句}

if 语句至多有 1 个 else 语句，else 语句在所有的 elseif 语句之后。

if 语句可以有若干个 elseif 语句，它们必须在 else 语句之前。

一旦其中一个 else if 语句检测为 true，其他的 else if 以及 else 语句都将跳过执行。

switch(expression){ case value : //语句 break;

case value : //语句 break; //可选 //你可以有任意数量的case语句 default : //可选 //语句 }

**switch case 执行时，一定会先进行匹配，匹配成功返回当前 case 的值，再根据是否有 break，判断是否继续输出，或是跳出判断.**

1. 循环

while( 布尔表达式 ) { //循环内容 }

只要布尔表达式为 true，循环就会一直执行下去

do { //代码语句}while(布尔表达式);

布尔表达式在循环体的后面，所以语句块在检测布尔表达式之前已经执行了。 如果布尔表达式的值为 true，则语句块一直执行，直到布尔表达式的值为 false

for(初始化; 布尔表达式; 更新) { //代码语句 }

如果为 true，循环体被执行。如果为false，循环终止，开始执行循环体后面的语句

for(声明语句 : 表达式) { //代码句子 }

break 主要用在循环语句或者 switch 语句中，用来跳出整个语句块

continue 适用于任何循环控制结构中。作用是让程序立刻跳转到下一次循环的迭代

1. 常用方法

Equals():判断number对象是否与参数相等

ceil():返回大于等于( >= )给定参数的的最小整数

floor():返回小于等于（<=）给定参数的最大整数

Round():四舍五入

Min():最小值

Max():最大值

Random():返回一个随机数

Length():返回字符串对象包含的字符数

charAt(index) 根据传入下标截取字符

toUpperCase() 转大写

toLowerCase() 转小写

split(str) 分割字符串

substring(beingindex,[endindex]) 截取字符串[含头不含尾]

currentTimeMillis() 获取1970年1月1日到现在的毫秒数

SimpleDateFormat sdf = new SimpleDateFormat(输出时间格式)

trim() 去首尾空格

1. 数组

dataType[] arrayRefVar=new dataType[arraySize];

dataType[] arrayRefVar = {value0, value1, ..., valuek};

type arrayName = new type[arraylenght1][arraylenght2];

1. IO流
2. 创建文件

public class CreateDir {

public static void main(String args[]) {

String dirname = "/tmp/user/java/bin";

File d = new File(dirname);

// 现在创建目录 d.mkdirs(); }

1. 区别

字节流和字符流的区别：

读写单位不同：字节流以字节（8bit）为单位，字符流以字符为单位，根据码表映射字符，一次可能读多个字节。

处理对象不同：字节流能处理所有类型的数据（如图片、avi等），而字符流只能处理字符类型的数据。

字节流：一次读入或读出是8位二进制。

字符流：一次读入或读出是16位二进制

#### ****只要是处理纯文本数据，就优先考虑使用字符流****

1. 事例

①读取文件内容

public String readFile( String filePath ){

FileInputStream fis=null; String result = "" ;

try { // 根据path路径实例化一个输入流的对象

fis = new FileInputStream( filePath );

//2. 返回这个输入流中可以被读的剩下的bytes字节的估计值；

int size = fis.available() ;

//3. 根据输入流中的字节数创建byte数组；

byte[] array = new byte[size];

//4.把数据读取到数组中；

fis.read( array ) ;

//5.根据获取到的Byte数组新建一个字符串，然后输出；

result = new String(array);

} catch (FileNotFoundException e) {

e.printStackTrace(); }

catch (IOException e) {

e.printStackTrace();

}finally{

if ( fis != null) {

try{

fis.close();

} catch (IOException e) {

e.printStackTrace(); } } }

return result ; }

②将内容写入文件

public void writeFile( String filePath , String content ){

FileOutputStream fos = null ;

try {

//1、根据文件路径创建输出流

fos = new FileOutputStream( filePath );

//2、把string转换为byte数组；

byte[] array = content.getBytes() ;

//3、把byte数组输出；

fos.write( array );

} catch (FileNotFoundException e) {

e.printStackTrace();

}catch (IOException e) { e.printStackTrace();

}finally{

if ( fos != null) {

try{

fos.close();

} catch (IOException e) {

e.printStackTrace(); } } } }

1. 面向对象
2. 封装

封装（英语：Encapsulation）是指一种将抽象性函式接口的实现细节部份包装、隐藏起来的方法

优点：

1. 良好的封装能够减少耦合。

2. 类内部的结构可以自由修改。

3. 可以对成员变量进行更精确的控制。

4. 隐藏信息，实现细节。

1. 继承

Java 不支持多继承，但支持多重继承

继承的特性

子类拥有父类非private的属性，方法。

子类可以拥有自己的属性和方法，即子类可以对父类进行扩展。

子类可以用自己的方式实现父类的方法。

Java的继承是单继承，但是可以多重继承，单继承就是一个子类只能继承一个父类，多重继承就是，例如A类继承B类，B类继承C类，所以按照关系就是C类是B类的父类，B类是A类的父类，这是java继承区别于C++继承的一个特性。

提高了类之间的耦合性（继承的缺点，耦合度高就会造成代码之间的联系越紧密，代码独立性越差）。

super关键字：我们可以通过super关键字来实现对父类成员的访问，用来引用当前对象的父类。

this关键字：指向自己的引用

final 关键字声明类可以把类定义为不能继承的

子类是不继承父类的构造器（构造方法或者构造函数）的，它只是调用（隐式或显式）

1. 多态

多态就是同一个接口，使用不同的实例而执行不同操作

多态存在的三个必要条件：继承、重写、父类引用指向子类对象

多态的实现方式：重写、接口、抽象类和抽象方法

1. 抽象

如果一个类中没有包含足够的信息来描绘一个具体的对象，这样的类就是抽象类

抽象类不能实例化对象

抽象类必须被继承，才能被使用

一个类只能继承一个抽象类，而一个类却可以实现多个接口

Abstract关键字同样可以用来声明抽象方法，抽象方法只包含一个方法名，而没有方法体

如果一个类包含抽象方法，那么该类必须是抽象类

任何子类必须重写父类的抽象方法，或者声明自身为抽象类

构造方法，类方法（用static修饰的方法）不能声明为抽象方法

1. 接口

接口是隐式抽象的，当声明一个接口的时候，不必使用**abstract**关键字

接口中每一个方法也是隐式抽象的，声明时同样不需要**abstract**关键字

接口中的方法都是公有的

类使用implements关键字实现接口

一个类只能继承一个类，但是能实现多个接口

抽象类和接口的区别

1. 抽象类中的方法可以有方法体，就是能实现方法的具体功能，但是接口中的方法不行。

2. 抽象类中的成员变量可以是各种类型的，而接口中的成员变量只能是 public static final 类型的。

3. 接口中不能含有静态代码块以及静态方法(用 static 修饰的方法)，而抽象类是可以有静态代码块和静态方法。

4. 一个类只能继承一个抽象类，而一个类却可以实现多个接口。

1. Java高级
2. 集合

List 允许存在null,允许重复 是一个有序集合

add() 新增元素

addAll(Collection) 将一个集合放入原集合的后面

get(index) 获取指定下标元素

set(index,object) 使用元素替换指定下标的原元素

isEmpty() 判断该集合是否为空

size() 获取集合的长度

remove() 可以根据指定下标或指定元素进行删除

clear() 清除集合中的所有元素

Vector 是一个非常古老的集合类 是线程安全的

ArrayList 是一个线程不安全，其底层仍是一个数组因此插入删除效率低，查询效率高

LinkedList 是一个线程不安全的 查询效率低 插入删除效率低

Set 不允许重复 最多包含一个null元素 它是一个无序的集合

常用方法：add(object)添加；remove(object)删除某个元素；isEmpty()判断是否为空；

Clear()清空所有内容

HashSet 是一个无序的 线程不安全的

TreeSet 是一个按照自然顺序进行排序的set集合 线程不安全的

LinkedHashSet 线程不安全的 是有序的

Map 键值对 key-value

常用方法：size()获取集合大小；values()获取集合所有的值；keyset()获取所有的键

Get()获取对应key的值；remove()根据key删除值；clear()清空所有

HashMap<泛型> 不允许重复，重复添加的值会覆盖之前的内容 线程不安全 无序

TreeMap 按照自然顺序排序 线程不安全

LinkedHashMap 有序的 线程不安全的

StringBuffer和StringBulider的比较：

1. 在执行速度方面的比较：StringBuilder > StringBuffer ；   
   　　 2、他们都是字符串变量，是可改变的对象，每当我们用它们对字符串做操作时，实际上是在一 个对象上操作的，不像String一样创建一些对象进行操作，所以速度快；   
   　  3、 StringBuilder：线程非安全的；   
   　　 4、StringBuffer：线程安全的；

对于String、StringBuffer和StringBulider三者使用的总结：   
　　 1.如果要操作少量的数据用 = String   
　 　2.单线程操作字符串缓冲区 下操作大量数据 = StringBuilder   
　　 3.多线程操作字符串缓冲区 下操作大量数据 = StringBuffer

1. 序列化

该类必须实现 java.io.Serializable 对象

该类的所有属性必须是可序列化的。如果有一个属性不是可序列化的，则该属性必须注明是短暂的

FileOutputStream fileOut = new FileOutputStream("/tmp/employee.ser"); ObjectOutputStream out = new ObjectOutputStream(fileOut);

out.writeObject(e);

out.close();

fileOut.close();

反序列化

FileInputStream fileIn = new FileInputStream("/tmp/employee.ser"); ObjectInputStream in = new ObjectInputStream(fileIn);

e = (Employee) in.readObject();

in.close();

fileIn.close();

1. 多线程

有五个状态：（1）新建状态

（2）就绪状态：调用start()方法后进入

（3）运行状态：如果就绪状态的线程获取 CPU 资源，就可以执行 **run()**

（4）阻塞状态：执行了sleep（睡眠）、suspend（挂起）等方法

（5）死亡状态：线程完成任务或者其他终止条件发生

创建线程的方法：①实现Runnable接口；②继承Tread类；③Callable和Future创建

while (true) {

//加上synchronized，并加入对象锁，new一个任意对象即可，我们这里使用Object来解决同步问题

synchronized (new Object()) {

if (ticket > 0) {

// 这里为了演示出线程不同步的问题，让线程睡眠一段时间，延时）

try {

Thread.sleep(1000L);

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println(saleWindowName + " 卖 出 了 " + ticket-- + " 号 票 ！");

} else {

break;

}

}

进程和线程的区别：

地址空间：同一进程的线程共享本进程的地址空间，而进程之间则是独立的地址空间。

资源拥有：同一进程内的线程共享本进程的资源如内存、I/O、cpu等，但是进程之间的资源是 独立的。

　　　　　 一个进程崩溃后，在保护模式下不会对其他进程产生影响，但是一个线程崩溃整个进 程都死掉。所以多进程要比多线程健壮。

　　　　　 进程切换时，消耗的资源大，效率高。所以涉及到频繁的切换时，使用线程要好于进 程。同样如果要求同时进行并且又要共享某些变量的并发操作，只能用线程不能用进 程

执行过程：每个独立的进程有一个程序运行的入口、顺序执行序列和程序出口。但是线程不能 独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。

线程是处理器调度的基本单位，但是进程不是。

两者均可并发执行。

对资源的管理和保护要求高，不限制开销和效率时，使用多进程

要求效率高，频繁切换时，资源的保护管理要求不是很高时，使用多线程

继承Thread类和Runnable的区别

避免了java单继承的局限性

如果多个线程操作同一数据（同一份资源），更适合使用Runnable接口

如果是Thread去操作同一数据则需要加上一个对象锁

而Runnable接口只需要synchronized(this){}

1. 反射

得到class的三种方式：

//1、通过对象调用 getClass() 方法来获取,通常应用在：比如你传过来一个 Object

// 类型的对象，而我不知道你具体是什么类，用这种方法

Person p1 = new Person();

Class c1 = p1.getClass();

//2、直接通过 类名.class 的方式得到,该方法最为安全可靠，程序性能更高

// 这说明任何一个类都有一个隐含的静态成员变量 class

Class c2 = Person.class;

//3、通过 Class 对象的 forName() 静态方法来获取，用的最多，

// 但可能抛出 ClassNotFoundException 异常12

Class c3 = Class.forName("com.ys.reflex.Person");

通过 Class 类获取成员变量、成员方法、接口、超类、构造方法等

getName()：获得类的完整名字。  
　　getFields()：获得类的public类型的属性。  
　　getDeclaredFields()：获得类的所有属性。包括private 声明的和继承类  
　　getMethods()：获得类的public类型的方法。  
　　getDeclaredMethods()：获得类的所有方法。包括private 声明的和继承类  
　　getMethod(String name, Class[] parameterTypes)：获得类的特定方法，name参数指定方法的名字，parameterTypes 参数指定方法的参数类型。  
　　getConstructors()：获得类的public类型的构造方法。  
　　getConstructor(Class[] parameterTypes)：获得类的特定构造方法，parameterTypes 参数指定构造方法的参数类型。  
　　newInstance()：通过类的不带参数的构造方法创建这个类的一个对象。

5.异常 Exception

检查性异常 是指 在编写过程当中被发现的错误

运行时异常 runtimeException 指的是在程序运行的过程当中发生的异常

nullpointexception

indexoutofboundsexception

numberformatexception

捕获异常

try{

}catch(Exception e){}

抛出异常

throw 抛出异常

如果一个方法当中抛出了一个异常，那么只有两种解决方式

一个是 throws 继续向上抛出，一直抛到顶层

第二种是 trycatch

注意事项

try不能既没有catch也没有finally

catch不能单独存在

try catch finally 之间不能存在其他代码

finally是可有可无的

1. mysql数据库

删除表

delete from 表名 删除全表的数据，不清除表结构

truncate 表名 删除表数据，清除表结构

drop table 表名 删除表

1. 图片上传

protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
 //获取项目存储上传文件位置的真实路径  
  String savePath = this.getServletContext().getRealPath("/upload");  
  //根据真实路径获取File对象  
  File file = new File(savePath);  
  //判断上传文件的保存目录是否存在  
  if (!file.exists() && !file.isDirectory()) {  
    System.out.println(savePath+"目录不存在，需要创建");  
       //创建目录  
       file.mkdir();}  
    //消息提示  
    String message = "";  
    int id = 0;  
    String filepath = "";  
    try{  
     //使用Apache文件上传组件处理文件上传步骤：  
     //1、创建一个DiskFileItemFactory工厂  
     DiskFileItemFactory factory = new DiskFileItemFactory();  
     //2、创建一个文件上传解析器  
     ServletFileUpload upload=new ServletFileUpload(factory);  
     //解决上传文件名的中文乱码  
     upload.setHeaderEncoding("UTF-8");   
     //3、判断提交上来的数据是否是上传表单的数据  
     if(!ServletFileUpload.isMultipartContent(req)){  
       //按照传统方式获取数据  
       return;}  
     //4、使用ServletFileUpload解析器解析上传数据，解析结果返回的是一个List<FileItem>集合，每一个FileItem对应一个Form表单的输入项  
     List<FileItem> list = upload.parseRequest(req);  
     for(FileItem item : list){  
       //如果fileitem中封装的是普通输入项的数据  
       if(item.isFormField()){  
         String name = item.getFieldName();  
         //解决普通输入项的数据的中文乱码问题  
         String value = item.getString("UTF-8");  
     /value = new String(value.getBytes("iso8859-1"),"UTF-8");  
         id = Integer.valueOf(value);  
         System.out.println(name + "=" + value);  
        }else{//如果fileitem中封装的是上传文件  
        //得到上传的文件名称，  
         String filename = item.getName();  
         System.out.println(filename);  
        if(filename==null || filename.trim().equals("")){  
             continue;}  
        //注意：不同的浏览器提交的文件名是不一样的，有些浏览器提交上来的文件名是带有路径的，如：  c:\a\b\1.txt，而有些只是单纯的文件名，如：1.txt  
      //处理获取到的上传文件的文件名的路径部分，只保留文件名部分  
      filename=filename.substring(filename.lastIndexOf("\\")+1);  
      //获取item中的上传文件的输入流  
      InputStream in = item.getInputStream();  
      //创建一个文件输出流  
      FileOutputStream out = new FileOutputStream(savePath + "\\" + filename);  
      filepath = "\\upload\\"+filename;  
      //创建一个缓冲区  
      byte buffer[] = new byte[1024];  
      //判断输入流中的数据是否已经读完的标识  
      int len = 0;  
      //循环将输入流读入到缓冲区当中，(len=in.read(buffer))>0就表示in里面还有数据  
      while((len=in.read(buffer))>0){  
     //使用FileOutputStream输出流将缓冲区的数据写入到指定的目录(savePath + "\\" + filename)当中  
     out.write(buffer, 0, len);}  
      //关闭输入流  
      in.close();  
      //关闭输出流  
      out.close();  
      //删除处理文件上传时生成的临时文件  
      item.delete();  
      message = "文件上传成功！"; } }  
     }catch (Exception e) {  
         message= "文件上传失败！";  
        e.printStackTrace();      
        }  
      //根据id找到书 修改书的信息  
      service.updateImgPath(id, filepath);  
      req.setAttribute("message",message);  
      req.getRequestDispatcher("/WEB-INF/index.jsp").forward(req, resp);  
}

下载：

HttpServletResponse resp) throws ServletException, IOException {

// 找到真实路径中文件位置并获取file对象

File f = new File(this.getServletContext().getRealPath(req.getParameter("path")));

// 打出输入流

FileInputStream fis = new FileInputStream(f);

// 防止乱码

String filename = URLEncoder.encode(f.getName(), "utf-8"); // 解决中文文件名下载后乱码的问题

byte[] b = new byte[fis.available()];

fis.read(b);

resp.setCharacterEncoding("utf-8");

// 下载设置

resp.setHeader("Content-Disposition", "attachment; filename=" + filename + "");

// 获取响应报文输出流对象

ServletOutputStream out = resp.getOutputStream();

// 输出

out.write(b);

out.flush();

out.close();

}

1. web前端

Css:

css使用的三种方式

1 外部样式表

<link rel="stylesheet" type="text/css" href="mystyle.css">

2 内部样式表

<style>

div {

width : 500px;

height: 600px;

background-color:black;

}

</style>

3 内联 直接在元素上进行定义

<div style="width:400px;height:500px;">

选择器

id选择器

# + id

class选择器

. + class

属性选择器

[属性]

标签选择器

标签 {}

子标签选择器

div p span a 以空格隔开

伪类： a:link {color:#FF0000;} /\* 未访问的链接 \*/

a:visited {color:#00FF00;} /\* 已访问的链接 \*/

a:hover {color:#FF00FF;} /\* 鼠标划过链接 \*/

a:active {color:#0000FF;} /\* 已选中的链接 \*/

1. Js以及ajax

弹窗：alert()警告框；confirm()确认框；prompt()提示框

$.ajax({

　　url:"/ajax\_get/",

　　type:"GET",

　　success:function(){})}

定时器：setInterval(对象，时间):设置定时器，clearTimeout(定时器):结束

2.Servlet

Servlet的生命周期： Servlet 通过调用 init () 方法进行初始化。

Servlet 调用 service() \doget()\dopost()方法来处理客户端的请求。

Servlet 通过调用 destroy() 方法终止（结束）。

最后，Servlet 是由 JVM 的垃圾回收器进行垃圾回收的

Servlet的过滤器：

<filter>

    <filter-name>LogFilter</filter-name>

    <filter-class>com.runoob.test.LogFilter</filter-class>

    <init-param>

        <param-name>Site</param-name>

        <param-value>菜鸟教程</param-value>

    </init-param>

    </filter>

Servlet的监听器：

<listener>

<listener-class>

com.journaldev.listener.AppContextListener

</listener-class></listener>

1. Jsp

Jsp的九大对象：out、request、response、session、application、

page、exception、pagecontext、config

Jsp的四大作用域：page、request、session、context

Jsp的三大指令：page指令、include指令、taglib指令

1. Java框架

事务：

四大特性：

①原子性：指事务包含的所有操作要么全部成功，要么全部失败回滚

②一致性：一个事务执行之前和执行之后都必须处于一致性状态

③隔离性：当多个用户并发访问数据库时，不能被其他事务的操作所干扰

④持久性：一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的

如果不考虑隔离性，会出现：

1. 脏读：指在一个事务处理过程里读取了另一个未提交的事务中的数据
2. 不可重复读：指在对于数据库中的某个数据，一个事务范围内多次查询却返回了不同的数据值
3. 虚读（幻读）：事务非独立执行时发生的一种现象

隔离级别：

① Serializable (串行化)：可避免脏读、不可重复读、幻读的发生。

② Repeatable read (可重复读)：可避免脏读、不可重复读的发生。

　③ Read committed (读已提交)：可避免脏读的发生。

　④ Read uncommitted (读未提交)：最低级别，任何情况都无法保证。

1. Hibernate

（1）Hibernate的执行流程：

1 编写hibernate.cfg.xml配置文件

2 使用Configuration加载/获取SessionFactory

3 使用SessionFactory获取session

4 session.beginTransaction 开始事务

5 执行业务逻辑/增删改查

6 提交事务 tx.commit()/tx.rollback()

7 关闭session session.close()

（2）hibernate的查询方式

导航查询

①oid => get/load

②hql

from

from 类名

Query query = session.createQuery(hql);

List objects = query.list();

select

select 列名 from 类名

String hql = select o.xxx from Object o;

Query query = session.createQuery(hql);

List objects = query.list();

from 默认的查询该类中的所有属性

select 可以指定查询该类的某个属性

分页

Session session = getSession();

String hql = "from User";

Query query = session.createQuery(hql);

//从第几条开始

query.setFirstResult(4);

//取多少条记录

query.setMaxResults(4);

List<User> users = query.list();

③criteria

Criteria

//使用criteria查询User

Criteria criteria = session.createCriteria(User.class);

//相当于加上 where id = 2

criteria.add(Restrictions.eq("id",2));

gt 大于

lt 小于

eq 等于

like 模糊查询

ilike 转小写之后模糊查询

between isnull isnotnull

...

组合条件

c1 c2 任意成立一个

LogicalExpression orExp = Restrictions.or(c1, c2);

c1 c2 两者同时成立

LogicalExpression andExp = Restrictions.and(c1, c2);

④sql

String sql = "select \* from user";

SQLQuery query = session.createSQLQuery(sql);

//建立user表和User类之间的联系

query.addEntity(User.class);

List<User> users = query.list();

（3）hibernate中整合ehcache

1 hibernate-ehcache ehcache-core slf4j-api

2 在hibernate.cfg.xml中配置

<!-- 缓存配置 -->

<!-- 打开二级缓存 默认是关闭的-->

<property name="hibernate.cache.use\_second\_level\_cache">true</property>

<!-- hibernate.cache.provider\_class hibernate4.x的写法-->

<!-- hibernate5.x的设置 -->

<!-- 设置缓存的提供商 -->

<property name="hibernate.cache.region.factory\_class">org.hibernate.cache.ehcache.internal.SingletonEhcacheRegionFactory</property>

<!-- 打开查询缓存 -->

<property name="hibernate.cache.use\_query\_cache">true</property>

3 新建ehcache.xml

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

<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://ehcache.org/ehcache.xsd"

updateCheck="false">

<!-- 设置硬盘目录 -->

<diskStore path="D:\\temp"/>

<!-- 默认缓存设置 -->

<!-- 设置在内存中的最大储存条数 -->

<!-- 是否永久保存 -->

<!-- 死亡周期时间 -->

<!-- 存活时间 -->

<!-- 是否覆盖 -->

<defaultCache

maxElementsInMemory="10000"

eternal="false"

timeToIdleSeconds="120"

timeToLiveSeconds="120"

overflowToDisk="true"/>

</ehcache>

4 设置哪些类需要缓存

4.1 在类中设置

4.1.1 在xml设置

<cache usage = "选择的缓存策略"/>

4.1.2 注解设置

@Cache(usage = "选择的缓存策略")

4.2 在hibernate.cfg.xml中设置

<class-cache class="类名" usage ="选择的缓存策略" />

1. Mybatis

一个持久层框架 ORM

Hibernate 与 Mybatis

hibernate是一个全自动 重量级的框架

mybatis是半自动 轻量级的框架

configuration => sqlsessionfactorybuilder => sqlsessionfactory => sqlSession

configuration => sessionfactory => session

mybatis的配置流程

1 配置mybatis的cfg文件

1.1 设置开发环境

1.2 设置事务管理的方式

1.3 设置数据源

1.4 加入类的映射文件

2 配置实体类的映射文件

2.1 先定义该映射的命名空间

2.2 添加sql语句

2.2.1 先选择需要的sql语句的类型

2.2.2 为这条语句定义id

2.2.3 如果有参数的话，设置参数的类型 paramterType

2.2.4 如果有返回的值的话，设置返回类型 resultType

2.2.5 如果有参数的话，sql里应该使用#{参数名} select \* from User where id = #{id}

3 获取sqlSession

4 选择需要执行的sql语句调用

mybatis配置文件

configuration 根节点

properties 引入配置文件

settings 对mybaits的全局设置

typeAliases 对类取一个别名

typeHandlers

objectFactory 对象工厂

plugins 插件

environments 配置环境

mappers 映射器

mybatis的映射文件

mapper namespace 命名空间

select/update/delete/insert

id 唯一标识符

paramterType 参数类型

resultType 返回类型

resultMap

jdbcType对应javaType

jdbctype javatype

VARCHAR String

SMALLINT short

INTEGER int

BIGINT long

DOUBLE double

REAL float

动态sql

if

choose

when

otherwise

where => 直接在sql语句中添加上一个 WHERE 关键字 => 不建议使用

set => 直接在sql语句中添加上一个 SET 关键字 => 不建议使用

trim => 智能where,set

prefix => 在sql中添加一个where

prefixOvrrides => 如果and/or 在where后面则自动取消掉 => 一般用于查询条件的链接

suffixOverrides => 如果,出现在后面 则自动取消 => 一般用于update语句

mybatis打印sql日志

1 新建一个log4j.properties

log4j.rootLogger=DEBUG, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

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

log4j.appender.stdout.layout.ConversionPattern=[service] %d - %c -%-4r [%t] %-5p %c %x - %m%n

#log4j.appender.R=org.apache.log4j.DailyRollingFileAppender

#log4j.appender.R.File=../logs/service.log

#log4j.appender.R.layout=org.apache.log4j.PatternLayout

#log4j.appender.R.layout.ConversionPattern=[service] %d - %c -%-4r [%t] %-5p %c %x - %m%n

#log4j.logger.com.ibatis = debug

#log4j.logger.com.ibatis.common.jdbc.SimpleDataSource = debug

#log4j.logger.com.ibatis.common.jdbc.ScriptRunner = debug

#log4j.logger.com.ibatis.sqlmap.engine.impl.SqlMapClientDelegate = debug

#log4j.logger.java.sql.Connection = debug

log4j.logger.java.sql.Statement = debug

log4j.logger.java.sql.PreparedStatement = debug

log4j.logger.java.sql.ResultSet =debug

2 在mybatis-configuration.xml

<settings>

<setting name="logImpl" value="STDOUT\_LOGGING"/>

<setting name="logImpl" value="LOG4J"/>

</settings>

resultMap

id property column

result property column

construtor 构造注入 使用构造注入时，是有顺序的

idArg 表示主键 在构造注入时，只需要表明列名和类型

Arg 表示其余列

<constructor>

<idArg column="id" javaType="int"></idArg>

<arg column="uname" javaType="String"></arg>

</constructor>

关联 一对一

association

① <association property="关联的对象名" column="关联的字段" javaType="对象类型">

被关联对象的字段映射

<id/>

<result/>

<association>

② <association property="关联的对象名" column="关联的字段" javaType="对象类型" select="查询该对象的sqlid"/>

<select id="xxx">查询被关联对象的信息</select>

③ <association property="关联的对象名" column="关联的字段" javaType="对象类型" resultMap="被关联对象的resultMap"/>

<select id="zzzz">将查询表和被关联表连表查询</select>

集合 一对多

collection

<collection property="sonList" column="id" javaType="ArrayList" ofType="model.PrivateSon" select="selectSonForUser"/>

javaType mybaits一般都会自动计算出类型，这个属性可以不写

ofType 表示的是这个集合的泛型

select 调用的查询语句

注解的步骤

1 新建一个XXXMaper

2 在mapper中书写resultmap以及增删改查

3 在mybatis\_config.xml中的mappers节点里写入 <package name = "mapper所在的包名">

4 从sqlsession调用getMapper(XXXMapper.class)对XXXMapper进行加载

5 从xxxmapper的实例进行方法的调用

@Results => 声明resultMap的注解，在里面可写入多个@Result,可以取一个名字 id = "xxx",方便以后的调用

@Result => 声明表中的列和实体类的属性的对应关系

@ResultMap => 使用@ResultMap(value = "xxx") 调用名字为xxx的@Results

1. Spring

特点

1 轻量级

2 依赖注入来实现低耦合 高聚合

3 面向切面/面向接口/面向函数

模块：core（核心）、dao、context、web、webmvc、orm、aop

两大特征：IOC 控制反转 AOP面向切面

IOC 控制反转

DI 依赖注入

1 构造注入 依赖构造方法

<constructor-arg name value/ref />

2 set注入 依赖set方法

name 参数名

value 参数值

ref 在Spring注册的bean的 id

在注册bean的时候，id和name的作用是一样 name可以写入特殊字符

Bean的生存周期

Scope

Singleton 默认为单例

prototype 每次都创建一个新的对象

request 在request的范围内

session 在session的范围内

application 在整个应用的范围内

当我们向Spring中使用@Component注入bean时，会自动将类名转成小写

注解部分

@Reopsitory => 标识dao层

@Service => 标识service层

@Controller => 标识controller层

@Component => 当不好分层的时候就统一使用它，Spring会统一识别为组件

AOP

面向切面编程，是对面向对象的有力补充

可以动态的添加和删除在切面上的逻辑而不影响原来的执行代码

Aspect 切面 Advice 通知 PointCut 切点

在哪个切点的哪个位置上放入切面

xml方式

Aspect

id -> 当前切面的id

ref -> 当前切面的依赖类

PointCut

id -> 当前切点的id

expression -> 当前使用该切点需满足表达式

Advice

around 环绕通知

before 前置通知

after 后置通知

after-throwing 异常通知

after-returnning 返回通知

method -> 调用切面的哪个方法

pointCut-ref -> 使用哪个切点

注解方式

@Aspect -> 标识在类或接口上

@PointCut(excuetion(切点表达式))

@Before/@After/@Around/@After-returnning..(切点的方法名)

在Spring的配置文件中加入

<aop:aspectj-autoproxy proxy-target-class="true"/>

环绕通知

public void around(ProceedingJoinPoint pip){

//执行完需要的操作后

return pip.proceed();

}

整合hibernate并创建事物管理

1 构建数据源 dbcp c3p0 druid jdbc

2 构建sessionFactory

2.1 依赖数据源

2.2 寻找映射文件/寻找注解类

2.3 关于hibernate的设置

3 创建hibernateTemplate -> 选择对应的版本

4 创建事物

4.1 创建事物管理器 transactionManager -> 选择对应的版本

4.2 使用aop 打开切面 -> 定义切点 -> 自定义transactionAdvice

4.3 声明事物的切入方式

1. SpringMVC

1 在web.xml中进行配置

<servlet>

<serlvet-name>xxx</servlet-name>

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

<!-- 如果没有指定去加载某个配置文件则默认寻找WEB-INF/${servlet-name}-servlet.xml -->

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

</servlet>

<servlet-mapping>

<serlvet-name>xxx</serlvet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

2 dispatcher-servlet.xml/springmvc.xml

<context:component-scan packge="控制层所在包名"/>

<mvc:annotation-driven/>

<!-- 视图解析器 -->

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<!-- 前缀 -->

<property name="prefix" value="/WEB-INF/views/" />

<!-- 后缀 -->

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

</bean>

3 对控制层做修改

1 首先标识控制层@Controller

2 添加请求的路径 @RequestMapping("xxxx") 可以标识在类上也可以标识方法上

3 获取参数，如果方法的形参和传入的实参名相同则可以直接获取到，如果不同则使用@RequestParam("实参名")获取参数 也可以直接获取对象

4 如果需要返回页面则需要在方法中设置返回值

返回

forward

forward:xxx => 将请求转发到指定的请求上,如果该请求含有一个父路由那么则转发含有父路由的请求中

forward:xxx.jsp => 转发到xxx.jsp上 如果该请求含有一个父路由那么则转发含有父路由的请求中

forward:/xxx.jsp => 转发到指定的jsp上

1. SSH

ssh整合步骤

1 applicationContext.xml

1 扫描包

2 引入db.properties

3 创建datasource

3.1 引入db.properties的属性值

3.2 配置连接池

4 创建sessionFactory

4.1 引入数据源

4.2 扫描包 => 加注解的实体类/实体类的映射文件

4.3 对hibernate的配置 => 设置方言／显示ｓｑｌ语句／格式化语句／hbm2ddl.auto

5 配置hibernatetemplate

5.1 引入sessionfactory

6 配置事务管理器

2 spring-mvc.xml

1 扫描controller

2 配置视图解析器

3 配置静态资源的映射

3 web.xml

1 设置contextparam

2 配置spring的监听器

3 配置dispatcherservlet