# 一、JAVA

## Io读取文件的几种方式

January 25, 2019

#### 总结

\*流里面的读写，建议使用BufferedInputStream、BufferedOutputStream

\*FileOutputStream有续写的能力，使用够造方法FileOutputStream(file, true)

#### InputStream、OutputStream

（字节流）

[复制代码](javascript:void(0);)

　　　　 //读取文件(字节流)

InputStream in = new FileInputStream("d:\\1.txt");

//写入相应的文件

OutputStream out = new FileOutputStream("d:\\2.txt");

//读取数据

//一次性取多少字节

byte[] bytes = new byte[2048];

//接受读取的内容(n就代表的相关数据，只不过是数字的形式)

int n = -1;

//循环取出数据

while ((n = in.read(bytes,0,bytes.length)) != -1) {

//转换成字符串

String str = new String(bytes,0,n,"GBK"); #这里可以实现字节到字符串的转换，比较实用

System.out.println(str);

//写入相关文件

out.write(bytes, 0, n);

}

//关闭流

in.close();

out.close();

#### BufferedInputStream、BufferedOutputStream

（缓存字节流）使用方式和字节流差不多，但是效率更高（推荐使用）

[复制代码](javascript:void(0);)

//读取文件(缓存字节流)

BufferedInputStream in = new BufferedInputStream(new FileInputStream("d:\\1.txt"));

//写入相应的文件

BufferedOutputStream out = new BufferedOutputStream(new FileOutputStream("d:\\2.txt"));

//读取数据

//一次性取多少字节

byte[] bytes = new byte[2048];

//接受读取的内容(n就代表的相关数据，只不过是数字的形式)

int n = -1;

//循环取出数据

while ((n = in.read(bytes,0,bytes.length)) != -1) {

//转换成字符串

String str = new String(bytes,0,n,"GBK");

System.out.println(str);

//写入相关文件

out.write(bytes, 0, n);

}

//清楚缓存

out.flush();

//关闭流

in.close();

out.close();

#### InputStreamReader、OutputStreamWriter

（字节流，这种方式不建议使用，不能直接字节长度读写）。使用范围用做字符转换

[复制代码](javascript:void(0);)

　　　　 //读取文件(字节流)

InputStreamReader in = new InputStreamReader(new FileInputStream("d:\\1.txt"),"GBK");

//写入相应的文件

OutputStreamWriter out = new OutputStreamWriter(new FileOutputStream("d:\\2.txt"));

//读取数据

//循环取出数据

byte[] bytes = new byte[1024];

int len = -1;

while ((len = in.read()) != -1) {

System.out.println(len);

//写入相关文件

out.write(len);

}

//清楚缓存

out.flush();

//关闭流

in.close();

out.close();

#### BufferedReader、BufferedWriter

(缓存流，提供readLine方法读取一行文本)

[复制代码](javascript:void(0);)

　　　　 //读取文件(字符流)

BufferedReader in = new BufferedReader(new InputStreamReader(new FileInputStream("d:\\1.txt"),"GBK"));#这里主要是涉及中文

//BufferedReader in = new BufferedReader(new FileReader("d:\\1.txt")));

//写入相应的文件

BufferedWriter out = new BufferedWriter(new OutputStreamWriter(new FileOutputStream("d:\\2.txt"),"GBK"));

//BufferedWriter out = new BufferedWriter(new FileWriter("d:\\2.txt"))；

//读取数据

//循环取出数据

String str = null;

while ((str = in.readLine()) != null) {

System.out.println(str);

//写入相关文件

out.write(str);

out.newLine();

}

//清楚缓存

out.flush();

//关闭流

in.close();

out.close();

#### Reader、PrintWriter

（PrintWriter这个很好用，在写数据的同时可以格式化）

[复制代码](javascript:void(0);)

　　　　 //读取文件(字节流)

Reader in = new InputStreamReader(new FileInputStream("d:\\1.txt"),"GBK");

//写入相应的文件

PrintWriter out = new PrintWriter(new FileWriter("d:\\2.txt"));

//读取数据

//循环取出数据

byte[] bytes = new byte[1024];

int len = -1;

while ((len = in.read()) != -1) {

System.out.println(len);

//写入相关文件

out.write(len);

}

//清楚缓存

out.flush();

//关闭流

in.close();

out.close();

## printf的用法

<https://www.cnblogs.com/seakt/p/4478045.html>

例 printf("%d, %o", a, a);

## 数据类型中的转换

中数据类型及其之间的转换

）int长度数据类型有：byte(8bits)、short(16bits)、int(32bits)、long(64bits)、

）float长度数据类型有：单精度（32bits float）、双精度（64bits double），JAVA中将

double类型，若要定义float需在数据后面用f声明；

）boolean类型变量的取值有：ture、false

）char数据类型有：unicode字符,16位

Integer、Float、Boolean、Character、Double、Short、Byte、Long

本的数据类型由低级到高级分别为：（byte、short、char）

int——long——float——double

：此处的“级别”是指表示值的范围的大小。

）低级到高级的自动类型转换；

）高级到低级的强制类型转换（会导致溢出或丢失精度）；

）基本类型向类类型转换；

）基本类型向字符串的转换；

）类类型向字符串转换

在一个双操作数以及位运算等算术运算式中，会根据操作数的类型将低级的数据类型自动

）只要两个操作数中有一个是double类型的，另一个将会被转换成double类型，并且

double类型；

）只要两个操作数中有一个是float类型的，另一个将会被转换成float类型，并且结果

float类型；

）只要两个操作数中有一个是long类型的，另一个将会被转换成long类型，并且结果也

long类型；

）两个操作数（包括byte、short、int、char）都将会被转换成int类型，并且结果也是

类型。

如果低级类型为char型，向高级类型（整型）转换时，会转换为对应ASCII码值，再做

对于byte,short,char三种类型而言，他们是平级的，因此不能相互自动转换，可以使用

如：

不能在布尔值和任何数字类型间强制类型转换；

不同级别数据类型间的强制转换，可能会导致溢出或精度的下降。

当字节类型变量参与运算，java作自动数据运算类型的提升，将其转换为int类型。例如：

b=3;

b=(byte)(b\*3);//必须声明byte。

单类型的变量转换为相应的包装类，可以利用包装类的构造函数。即：

、Character(char value)、Integer(int value)、Long(long value)、

、Double(double value)

包装类中，总有形为××Value()的方法，来得到其对应的简单类型数据。利用这种

也可以实现不同数值型变量间的转换，例如，对于一个双精度实型类，intValue()可

doubleValue()可以得到其对应的双精度实型变量。

．字符串与其它类型间的转换

其它类型向字符串的转换

调用类的串转换方法:X.toString();

自动转换:X+“”;

使用String的方法:String.volueOf(X);

字符串作为值,向其它类型的转换

先转换成相应的封装器实例,再调用对应的方法转换成其它类型

字符中“32.1”转换double型的值的格式为:new Float(“32.1”).doubleValue()。也可以

:Double.valueOf(“32.1”).doubleValue()

静态parseXXX方法

.parseInt( s );

Character的getNumericValue(char ch)方法

api。

）基本类型向类类型转换

new出一个新的类类型的变量

>基本数据类型

.parseInt(“123”)

“123”).floatValue()

Float对象，然后再调用这个对象的floatValue()

float数值。

“123”).booleanValue()

明：上例是将一个字符串转化成一个Boolean对象，然后再调用这个对象的

方法返回其对应的boolean数值。

“123”).doubleValue()

上例是将一个字符串转化成一个Double对象，然后再调用这个对象的doubleValue()

double数值。

“123”).longValue()

Long对象，然后再调用这个对象的longValue()

long数值。

.valueOf(“123”).charValue()

明：上例是将一个字符串转化成一个Character对象，然后再调用这个对象的

方法返回其对应的char数值。

）基本类型向字符串的转换

// “""”把2转成字符串操作；

// 不存在转换。

// 前两个数值相加后，被“""”转成字符串。

// 同第一个。

23，5，5，23

）类类型向字符串转换

object类的子类，而所有的object类都有一个toString()函

toString()函数来转换即可

new出一个新的类类型的变量

.valueOf(“123”).intValue()

Integer对象，然后再调用这个对象的intValue()

int数值。

“123”).floatValue()

Float对象，然后再调用这个对象的floatValue()

float数值。

“123”).booleanValue()

明：上例是将一个字符串转化成一个Boolean对象，然后再调用这个对象的

方法返回其对应的boolean数值。

“123”).doubleValue()

上例是将一个字符串转化成一个Double对象，然后再调用这个对象的doubleValue()

double数值。

“123”).longValue()

Long对象，然后再调用这个对象的longValue()

long数值。

.valueOf(“123”).charValue()

明：上例是将一个字符串转化成一个Character对象，然后再调用这个对象的

方法返回其对应的char数值。

## 反射机制

****getModifiers()****

PUBLIC: 1

PRIVATE: 2

PROTECTED: 4

STATIC: 8

FINAL: 16

SYNCHRONIZED: 32

VOLATILE: 64

TRANSIENT: 128

NATIVE: 256

INTERFACE: 512

ABSTRACT: 1024

STRICT: 2048

**getDeclaredFields()**

获得某个类的所有声明的字段，即包括public、private和proteced，但是不包括父类的申明字段。

属性的操作案例

Class<?> clazz = target.getClass();// 获取PrivateClass整个类

// Object pc = clazz.newInstance();// 创建一个实例

Field[] fs = clazz.getDeclaredFields();// 获得某个类的所有声明的字段，即包括public、private和proteced，但是不包括父类的申明字段。

Field field;Object value;

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

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

(field = fs[i]).setAccessible(true);// 将目标属性设置为可以访问

if((value = field.get(target)) != null

&& field.getModifiers() == 2) // PRIVATE: 2

map.put(field.getName(), value);

}

## 内部类

### 作用

* 如果你想实现一个接口，但是这个接口中的一个方法和你构想的这个类中的一个方法的名称，参数相同。这时候，你可以建一个内部类实现这个接口
* 很好的实现多继承的效果。
* 你甚至连这两个内部类的名字都没有看见！这样，内部类的第一个好处就体现出来了 隐藏你不想让别人知道的操作，也即封装性
* 非静态内部类对象有着指向其外部类对象的引用。即：一个内部类对象可以访问创建它的外部类对象的内容，甚至包括私有变量！这是一个非常有用的特性

### 创建方式

public class Test0530 {

private int valueRateRate = 2;

private class Content implements Contents {

private int i = 11;

public int value() {

// 此处证明作用第四点（非静态内部类对象有着指向其外部类对象的引用）

// return i \* valueRateRate;

// 如果内外两个类变量名相同 还可以这样区分引用外部类的变量

return i \* Test0530.this.valueRateRate;

}

}

protected class GDestination implements Destination {

private String label;

private GDestination(String whereTo) {

label = whereTo;

}

public String readLabel() {

return label;

}

}

public Destination dest(String s) {

return new GDestination(s);

}

public Contents cont() {

return new Content();

}

public static void main(String[] args) {

Test0530 p = new Test0530();

// 创建内部类的第一种方式：证明了作用中的第三点（隐蔽性）

/\*Contents c = p.cont();

Destination d = p.dest("Beijing"); \*/

// 创建内部类的第二种方式：此时创建的话需要实例化的外部类对象，

// 为什呢？这里可以看作用的第四点（非静态内部类对象有着指向其外部类对象的引用）

Test0530.GDestination dest = p.new GDestination("Beijing");

}

}

interface Contents {

int value();

}

interface Destination {

String readLabel();

}

### 静态内部类

**区别就在于静态内部类没有了指向外部的引用**

两者区别：非静态内部类中，都不能有静态数据，静态方法或者又一个静态内部类（内部类的嵌套可以不止一层）。不过静态内部类中却可以拥有这一切。

### 局部内部类

从下面代码中可以看出局部内部类可以写在某个方法里，除了这种还可以在方法中的if语句块中进行编写内部类代码，创建方式多种多样，具体作用具体分析

public Destination dest(String s) {

class GDestination implements Destination {

private String label;

private GDestination(String whereTo) {

label = whereTo;

}

public String readLabel() {

return label;

}

}

return new GDestination(s);

}

### 匿名内部类

当你只需要创建一个类的对象而且用不上它的名字时，可以使用这种方式new interfacename(){......}; 或 new superclassname(){......};，如下具体方式使代码看上去简洁清楚。

public class Goods3 {

public Contents cont() {

return new Contents() {

private int i = 11;

public int value() {

return i;

}

};

}

}

## 集合

### Lsit的遍历

List<Object> lists =new ArrayList<Object>() ;







### Map的遍历

Map<String, Object> maps = new HashMap<String, Object>() ;

for(Map.Entry<String , Object> entry : maps.entrySet()){

System.out.println(entry.key + entry.value) ;

}

#### 对于HashMap 类中 put（）的解析：

**HashMap 类中的：**

static final int hash(Object key){

int h ;

//无符号右移的规则只记住一点：忽略了符号位扩展，0补最高位  无符号右移运算符>>> 只是对32位和64位的值有意义

//<<      :     左移运算符，num << 1,相当于num乘以2

//>>      :     右移运算符，num >> 1,相当于num除以2

//>>>    :     无符号右移，忽略符号位，空位都以0补齐

return (key == null ) ? 0 (h = key.hashCode()) ^ (h >>> 16)

}

public final int hashCode(){

return Object.hashCode(key) ^ Objects.hashCode(value) ;

}

**Object类中hashCode(Object o)**

返回非@code空参数的散列码和0

public static int hashCode(Object o){

return o != null ? o.hashCode() : 0 ;

}

node<K,V> newNode(int hash, K key, V value, Node<K,V> next) {

return new Node<>(hash, key, value, next);

}

### Hash表 **哈希表**

#### **哈希表**

哈希表（Hash table，也叫散列表），是根据关键码值(Key value)而直接进行访问的**[数据结构](http://lib.csdn.net/base/datastructure" \t "_blank" \o "算法与数据结构知识库)**。也就是说，它通过把关键码值映射到表中一个位置来访问记录，以加快查找的速度。这个映射函数叫做散列函数，存放记录的数组叫做散列表。

记录的存储位置=f(关键字)

这里的对应关系f称为散列函数，又称为哈希（Hash函数），采用散列技术将记录存储在一块连续的存储空间中，这块连续存储空间称为散列表或哈希表（Hash table）。

哈希表hashtable(key，value) 就是把Key通过一个固定的**[算法](http://lib.csdn.net/base/datastructure" \t "_blank" \o "算法与数据结构知识库)**函数既所谓的哈希函数转换成一个整型数字，然后就将该数字对数组长度进行取余，取余结果就当作数组的下标，将value存储在以该数字为下标的数组空间里。（或者：把任意长度的输入（又叫做预映射， pre-image），通过散列算法，变换成固定长度的输出，该输出就是散列值。这种转换是一种压缩映射，也就是，散列值的空间通常远小于输入的空间，不同的输入可能会散列成相同的输出，而不可能从散列值来唯一的确定输入值。简单的说就是一种将任意长度的消息压缩到某一固定长度的消息摘要的函数。）  
    而当使用哈希表进行查询的时候，就是再次使用哈希函数将key转换为对应的数组下标，并定位到该空间获取value，如此一来，就可以充分利用到数组的定位性能进行数据定位。

数组的特点是：寻址容易，插入和删除困难；

而链表的特点是：寻址困难，插入和删除容易。

Hash的应用

Hash主要用于信息安全领域中加密算法，它把一些不同长度的信息转化成杂乱的128位的编码,这些编码值叫做Hash值.

2、查找：哈希表，又称为散列，是一种更加快捷的查找技术。

3、Hash表在海量数据处理中有着广泛应用

散列法

除法散列法

平方散列法

斐波那契（Fibonacci）散列法

# 二、日志的使用

通过测试以下代码又在MyBlog(ssm)/springboot项目中使用可行

**相关代码：**

// 所在包

import org.apache.log4j.Logger;

// 创建Logger对象

private static Logger log = Logger.getLogger(MainControl.class); // 日志对象

// 操作日志测试

log.info("你好"); // 打印

log.debug("你好"); // 未打印

log.warn("你好"); // 打印

log.error("你好"); // 打印

**Springboot中的依赖：**

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

**log4j.properties文件：**

#log4j.rootLogger=ERROR

log4j.rootLogger=ERROR,console,chen

log4j.logger.org.hibernate.cache=error

log4j.appender.console=org.apache.log4j.ConsoleAppender

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

log4j.appender.console.layout.ConversionPattern=[%-5p] %d{HH\:mm\:ss\:S} [%C{1}.%M\:%L] %m %n

log4j.logger.freemarker=ERROR

log4j.logger.com.chen=INFO

log4j.logger.net.sf.ehcache=ERROR

log4j.logger.org.apache.shiro=ERROR

log4j.logger.org.springframework.web=ERROR

log4j.logger.net.sf.hibernate=ERROR

log4j.logger.org.hibernate.tool.hbm2ddl=INFO

log4j.logger.org.hibernate.SQL=ERROR

#\u4e3a\u4e86\u663e\u793a\u53c2\u6570TRACE

log4j.logger.org.hibernate.type.descriptor.sql.BasicBinder=ERROR

log4j.logger.org.hibernate.type.descriptor.sql.BasicExtractor=ERROR

#\u67e5\u770b\u67e5\u8be2\u4e2d\u547d\u540d\u53c2\u6570\u7684\u503c

log4j.logger.org.hibernate.engine.QueryParameters=ERROR

log4j.logger.org.hibernate.engine.query.HQLQueryPlan=ERROR

#log4j.appender.stdout.Encoding=GBK

log4j.appender.chen=org.apache.log4j.DailyRollingFileAppender

log4j.appender.chen.File=system/springboottest/chen.log

log4j.appender.chen.immediateFlush=true

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

log4j.appender.chen.layout.ConversionPattern=[%-5p] %d{HH:mm:ss:S} [%C{1}.%M:%L] %m %n

# 三、Mybatis

## 1.简介

是一款优秀的持久层框架，它支持定制化 SQL、存储过程以及高级映射。

## 2.核心API

**SqlSessionFactory/SqlSessionFactoryBuilder/SqlSession**

SqlSessionFactory sessionFactory = new SqlSessionFactoryBuilder()

SqlSession session = sessionFactory.openSession() ;

HotelMapper hotelMapper = session.getMapper(HotelMapper.class);

## 3.动态查询

mybatis 的动态sql语句是基于OGNL表达式的。可以方便的在 sql 语句中实现某些逻辑. 总体说来mybatis 动态SQL 语句主要有以下几类:

### if 语句 (简单的条件判断)

<select id="getEmpsByConditionIf" resultType="com.atguigu.mybatis.bean.Employee">

select \* from tbl\_employee

<!-- where -->

<where>

<!-- test：判断表达式（OGNL）

OGNL参照PPT或者官方文档。

c:if test

从参数中取值进行判断

遇见特殊符号应该去写转义字符：

&&：

-->

<if test="id!=null">

id=#{id}

</if>

<if test="lastName!=null and lastName!=''">

and last\_name like #{lastName}

</if>

<if test="email!=null and email.trim()!=''">

and email=#{email}

</if>

<!-- ognl会进行字符串与数字的转换判断 "0"==0 -->

<if test="gender==0 or gender==1">

and gender=#{gender}

</if>

</where>

</select>

### choose (when,otherwize)

,相当于java 语言中的 switch ,与 jstl 中的choose 很类似

分支选择；带了break的swtich-case

如果带了id就用id查，如果带了lastName就用lastName查;只会进入其中一个

### Trim

(对包含的内容加上 prefix,或者 suffix 等，前缀，后缀)

字符串截取(where(封装查询条件), set(封装修改条件))

<!-- public List<Employee> getEmpsByConditionTrim(Employee employee); -->

<select id="getEmpsByConditionTrim" resultType="com.atguigu.mybatis.bean.Employee">

select \* from tbl\_employee

<!-- 后面多出的and或者or where标签不能解决

prefix="":前缀：trim标签体中是整个字符串拼串 后的结果。

prefix给拼串后的整个字符串加一个前缀

prefixOverrides="":

前缀覆盖： 去掉整个字符串前面多余的字符

suffix="":后缀

suffix给拼串后的整个字符串加一个后缀

suffixOverrides=""

后缀覆盖：去掉整个字符串后面多余的字符

-->

<trim prefix="where" suffixOverrides="and">

<if test="id != null">

id=#{id}

</if>

<if test="lastName != null and lastName != ''">

and last\_name like #{lastName}

</if>

<if test="email != null">

and email=#{email}

</if>

<if test="gender ==0 or gender ==1">

and gender=#{gender}

</if>

</trim>

### where

(主要是用来简化sql语句中where条件判断的，能智能的处理 and or ,不必担心多余导致语法错误)

<!-- public List<Employee> getEmpsByConditionChoose(Employee employee); -->

<select id="getEmpsByConditionChoose" resultType="com.atguigu.mybatis.bean.Employee">

select \* from tbl\_employee

<where>

<choose>

<when test=" id != null">

id like #{id}

</when>

<when test="lastName != null">

last\_name like #{lastName}

</when>

<when test="email != null">

email like #{email}

</when>

<otherwise>

gender = 0

</otherwise>

</choose>

</where>

</select>

### set (主要用于更新时)

<!-- public void updateEmp(Employee employee); -->

<update id="updateEmp">

update tbl\_employee

<set>

<if test="lastName!=null">

last\_name=#{lastName},

</if>

<if test="email!=null">

email=#{email},

</if>

<if test="gender!=null">

gender=#{gender}

</if>

</set>

where id=#{id}

</update>

1. foreach (在实现 mybatis in 语句查询时特别有用)

遍历集合

<!-- public List<Employee> getEmpsByConditionForeach(@Param("ids")List<Integer> ids);-->

<select id="getEmpsByConditionForeach" resultType="com....Employee">

<!--

collection：指定要遍历的集合：

list类型的参数会特殊处理封装在map中，map的key就叫list

item：将当前遍历出的元素赋值给指定的变量

separator:每个元素之间的分隔符

open：遍历出所有结果拼接一个开始的字符

close:遍历出所有结果拼接一个结束的字符

index:索引。遍历list的时候是index就是索引，item就是当前值

遍历map的时候index表示的就是map的key，item就是map的值

#{变量名}就能取出变量的值也就是当前遍历出的元素

-->

select \* from tbl\_employee

<foreach collection="ids" item="item\_id" separator=","

open="where id in(" close=")">

#{item\_id}

</foreach>

</select>

## 4.映射文件

### 4.1#{}、${}的区别

1. **#{}：**

PreparedStatement ps = conn.prepareStatement(sql);

ps.setInt(1,id);

优点：更安全，更迅速，通常也是首选做法。防止sql注入攻击

1. **${}：**

Statement st = conn.createStatement();

ResultSet rs = st.executeQuery(sql);

缺点：以这种方式接受从用户输出的内容并提供给语句中不变的字符串是不安全的，会导致潜在的 SQL 注入攻击，因此要么不允许用户输入这些字段，要么自行转义并检验。

### 4.2jdbcType类型归纳

Mybatis中javaType和jdbcType对应关系：

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

JDBC Type Java Type

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

CHAR String

VARCHAR String

LONGVARCHAR String

NUMERIC java.math.BigDecimal

DECIMAL java.math.BigDecimal

BIT boolean

BOOLEAN boolean

TINYINT byte

SMALLINT short

INTEGER int

BIGINT long

REAL float

FLOAT double

DOUBLE double

BINARY byte[]

VARBINARY byte[]

LONGVARBINARY byte[]

DATE java.sql.Date

TIME java.sql.Time

TIMESTAMP java.sql.Timestamp

CLOB Clob

BLOB Blob

ARRAY Array

DISTINCT mapping of underlying type

STRUCT Struct

REF Ref

DATALINK java.net.URL[color=red][/color]

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

例子：

Mybatis中javaType和jdbcType对应和CRUD例子：

<resultMap type="java.util.Map" id="resultjcm">

<result property="FLD\_NUMBER" column="FLD\_NUMBER" javaType="double" jdbcType="NUMERIC"/>

<result property="FLD\_VARCHAR" column="FLD\_VARCHAR" javaType="string" jdbcType="VARCHAR"/>

<result property="FLD\_DATE" column="FLD\_DATE" javaType="java.sql.Date" jdbcType="DATE"/>

<result property="FLD\_INTEGER" column="FLD\_INTEGER" javaType="int" jdbcType="INTEGER"/>

<result property="FLD\_DOUBLE" column="FLD\_DOUBLE" javaType="double" jdbcType="DOUBLE"/>

<result property="FLD\_LONG" column="FLD\_LONG" javaType="long" jdbcType="INTEGER"/>

<result property="FLD\_CHAR" column="FLD\_CHAR" javaType="string" jdbcType="CHAR"/>

<result property="FLD\_BLOB" column="FLD\_BLOB" javaType="[B" jdbcType="BLOB" />

<result property="FLD\_CLOB" column="FLD\_CLOB" javaType="string" jdbcType="CLOB"/>

<result property="FLD\_FLOAT" column="FLD\_FLOAT" javaType="float" jdbcType="FLOAT"/>

<result property="FLD\_TIMESTAMP" column="FLD\_TIMESTAMP" javaType="java.sql.Timestamp" jdbcType="TIMESTAMP"/>

</resultMap>

### 4.3resultType 返回类型：

**javaBean ：**

java代码：

Employee getEmpById(Integer id);

配置：

<select id="getEmpById" resultType="employee">

**List<T> ：**

java代码：

List<Employee> getAllEmps();

配置：

<select id="getAllEmps" resultType="employee"> // 返回值类型是集合内存储数据的类型，不是 'list'

**Map<String, Object> ：**

查询结果为一条：

Map中的数据格式为{表字段名， 对应的值}

java代码：

Map<String, Object> getEmpAsMapById(Integer id);

配置：

<select id="getEmpAsMapById" resultType="map">

查询结果为多条：

Map中的数据格式为{表字段名， 对应的值}

java代码：

Map<String, Object> getEmpAsMapById(Integer id);

配置：

<select id="getEmpAsMapById" resultType="map">

**自定义返回一个对象中的那些字段：**

使用心得

在多表查询中可能内心会问，此处配置了关联，对面需要配置关联吗。答案是不需要

java代码：

List<DdChargeWater> gets();

配置：

// resultMap 内容为空 按给定类型DdChargeWater中的属性映射

<resultMap id="userList" type="com.blog.entity.DdChargeWater"> </resultMap>

/\*

在resultMap 中自定义映射字段属性，这多用于多表查询

<resultMap id="userList" type="com.blog.entity.DdChargeWater">

<id column="表的主键字段，或者可以为查询语句中的别名字段" jdbcType="字段类型" property="映射pojo对象的主键属性" />

<association property="pojo的一个对象属性" javaType="pojo关联的pojo对象">

<id column="关联pojo对象对应表的主键字段" jdbcType="字段类型" property="关联pojo对象的主席属性"/>

<result column="任意表的字段" jdbcType="字段类型" property="关联pojo对象的属性"/>

</association>

<!-- 集合中的property须为oftype定义的pojo对象的属性-->

<collection property="pojo的集合属性" ofType="集合中的pojo对象">

<id column="集合中pojo对象对应的表的主键字段" jdbcType="字段类型" property="集合中pojo对象的主键属性" />

<result column="可以为任意表的字段" jdbcType="字段类型" property="集合中的pojo对象的属性" />

</collection>

</resultMap>

\*/

<select id="gets" resultMap="userList" useCache="false">

SELECT \* FROM ${table}

</select>

**以及返回Boolean，Integer 等；**

## 5.注解的使用

### @Select

### @ResultType

使用该注解来映射查询结果到实体类属性。

@ResultType(Demo.class)

### @Results的总结

MyBatis中使用@Results注解来映射查询结果集到实体类属性。

#### （1）@Results的基本用法。

当数据库字段名与实体类对应的属性名不一致时，可以使用@Results映射来将其对应起来。column为数据库字段名，porperty为实体类属性名，jdbcType为数据库字段数据类型，id为是否为主键。

@Select({"select id, name, class\_id from my\_student"})

@Results({

@Result(column="id", property="id", jdbcType=JdbcType.INTEGER, id=true),

@Result(column="name", property="name", jdbcType=JdbcType.VARCHAR),

@Result(column="class\_id ", property="classId", jdbcType=JdbcType.INTEGER)

})

List<Student> selectAll();

如上所示的数据库字段名class\_id与实体类属性名classId，就通过这种方式建立了映射关系。

#### @ResultMap的用法。

当这段@Results代码需要在多个方法用到时，为了提高代码复用性，我们可以为这个@Results注解设置id，然后使用@ResultMap注解来复用这段代码。

@Select({"select id, name, class\_id from my\_student"})

@Results(id="studentMap", value={

@Result(column="id", property="id", jdbcType=JdbcType.INTEGER, id=true),

@Result(column="name", property="name", jdbcType=JdbcType.VARCHAR),

@Result(column="class\_id ", property="classId", jdbcType=JdbcType.INTEGER)

})

List<Student> selectAll();

@Select({"select id, name, class\_id from my\_student where id = #{id}"})

@ResultMap(value="studentMap")

Student selectById(integer id);

#### @One的用法。

当我们需要通过查询到的一个字段值作为参数，去执行另外一个方法来查询关联的内容，而且两者是一对一关系时，可以使用@One注解来便捷的实现。比如当我们需要查询学生信息以及其所属班级信息时，需要以查询到的class\_id为参数，来执行ClassesMapper中的selectById方法，从而获得学生所属的班级信息。可以使用如下代码。

@Select({"select id, name, class\_id from my\_student"})

@Results(id="studentMap", value={

@Result(column="id", property="id", jdbcType=JdbcType.INTEGER, id=true),

@Result(column="name", property="name", jdbcType=JdbcType.VARCHAR),

@Result(column="class\_id ", property="myClass", javaType=MyClass.class,

one=@One(select="com.my.mybatis.mapper.MyClassMapper.selectById"))

})

List<Student> selectAllAndClassMsg();

#### @Many的用法。

与@One类似，只不过如果使用@One查询到的结果是多行，会抛出TooManyResultException异常，这种时候应该使用的是@Many注解，实现一对多的查询。比如在需要查询学生信息和每次考试的成绩信息时。

@Select({"select id, name, class\_id from my\_student"})

@Results(id="studentMap", value={

@Result(column="id", property="id", jdbcType=JdbcType.INTEGER, id=true),

@Result(column="name", property="name", jdbcType=JdbcType.VARCHAR),

@Result(column="class\_id ", property="classId", jdbcType=JdbcType.INTEGER),

@Result(column="id", property="gradeList", javaType=List.class,

many=@Many(select="com.my.mybatis.mapper.GradeMapper.selectByStudentId"))

})

List<Student> selectAllAndGrade();

## 6.多表查询

### 参考一

一对一：

**方法一：**

此处的UserExt是User的子类，它将包含所查询的所有属性。与查询结果对应

<select id="queryUserAndCellphone" resultType="com.bamzhy.bean.UserExt">

SELECT \* FROM student LEFT OUTER JOIN cellphone ON student.id = cellphone.sid

</select>

@Test

public void test11(){

ArrayList<UserExt> userExt = dao.queryUserAndCellphone();

System.out.println("userext= "+userExt);

}

**方法二：**

使用ResultMap

<resultMap id="usersandCellphone" type="user">

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

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

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

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

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

<!--Map里边的名为property的javaType类型引用使用select方法以column作为参数查找-->

<association property="cellphone"

javaType="com.bamzhy.bean.Cellphone"

select="com.bamzhy.dao.UserDao.queryCellphoneBySid"

column="id">

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

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

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

</association>

</resultMap>

<select id="queryUserAndCellphone2" resultMap="usersandCellphone" >

select \* from student

</select>

<select id="queryCellphoneBySid" parameterType="int" resultType="com.bamzhy.bean.Cellphone">

select \* from cellphone where sid = #{id}

</select>

一对多：

<resultMap id="userAndCellphones" type="user">

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

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

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

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

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

<collection property="cellphones"

javaType="list"

select="com.bamzhy.dao.UserDao.queryCellphoneBySid"

column="id">

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

</collection>

</resultMap>

<select id="queryUserById" parameterType="int" resultMap="userAndCellphones" >

select \* from student WHERE id = #{id}

</select>

<select id="queryCellphoneBySid" parameterType="int" resultType="com.bamzhy.bean.Cellphone">

select \* from cellphone where sid = #{id}

</select>

多对多：

<resultMap id="studentAndCourses" type="com.bamzhy.bean.Student">

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

<collection property="courses"

javaType="list"

select="com.bamzhy.dao.UserDao.findcourseBySid"

column="id">

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

</collection>

</resultMap>

<select id="findStudentByID" parameterType="int" resultMap="studentAndCourses" >

select \* from student WHERE id = #{id}

</select>

<select id="findcourseBySid" parameterType="int" resultType="com.bamzhy.bean.Course">

select \* from t\_sc INNER JOIN course on t\_sc.cid = course.cid and sid = #{sid}

</select>

### 参考二

<https://www.cnblogs.com/xdp-gacl/p/4264440.html>

## 7.MySql注入

<https://www.cnblogs.com/ichunqiu/p/9604564.html>

# 项目记录

## shopManerge

### 1.1整合Thymeleaf

初步学习springboot

测试th:href 对静态文件的访问，以及链接的请求 （th:href请求指向static和templates

mycss.css

login.html

register.html

LoginController.java

login使用a标签向register页面跳转<a th:href="@{register}">注册</a><br>

login对静态文件的获取mycss.css<link rel="stylesheet" th:href="@{css/mycss.css}" />

### 1.2初步搭建

login.html

LoginController.java

依赖模块Thymeleaf

## Jsptest/springboottest

### 2.1整合jsp

#### 2.1.1依赖配置

<!-- servlet 依赖 -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

</dependency>

<!-- JSTL jsp标准标签库 依赖 -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

<!-- tomcat 依赖 -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

</dependency>

<!-- jsp引擎 依赖 -->

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

</dependency>

#### 2.1.2application.yml配置文件

在application中指向jsp文件路径 /WEB-INF/jsp/ 和 后缀.jsp

spring:

datasource:

username: chenjiwei

password: cjw168999

url: jdbc:mysql://120.79.149.233:3306/shop

driver-class-name: com.mysql.jdbc.Driver

hikari:

idle-timeout: 60000

maximum-pool-size: 30

minimum-idle: 10

mvc:

view:

prefix: /WEB-INF/jsp/

suffix: .jsp

#### 2.1.3成功访问：

http://localhost:8080/login

### 2.2在项目中使用fastjson替换原来默认的jackson

Controller返回的是一个java对象或者是list集合时自动转换为json数据

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.2.29</version>

</dependency>

### 2.3日志的使用

具体见一、JAVA 日志的使用

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

### 2.4热部署

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<!-- optional=true,依赖不会传递 -->

<optional>true</optional>

<scope>true</scope>

</dependency>

<!-- 必须含有build标签的内容，不然热部署失效 -->

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<fork>true</fork>

</configuration>

</plugin>

</plugins>

</build>

### 2.5单元测试

Springboot中的单元测试，其实就是模拟请求，测试请求的数据是否正确

#### 2.5.1依赖配置

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

#### 2.5.2代码：

**LoginControlTest.java**

@RunWith(SpringRunner.class)

@SpringBootTest(classes = App.class)

public class LoginControlTest {

// 注入spring容器

@Autowired

private WebApplicationContext wac;

// 实现了对Http请求的模拟

private MockMvc mvc;

@Before

public void setupMockMvc(){

mvc = MockMvcBuilders.webAppContextSetup(wac).build();

}

@Test

@Transactional

public void getUserInfor(){

// MediaType.APPLICATION\_FORM\_URLENCODED

// Content-Type:"application/x-www-form-urlencoded"

String data="loginName=root";

try {

mvc.perform(MockMvcRequestBuilders.post("/user")

.contentType(MediaType.APPLICATION\_FORM\_URLENCODED)

.accept(MediaType.APPLICATION\_JSON\_UTF8)

.content(data)

).andExpect(MockMvcResultMatchers.status().isOk())

.andDo(MockMvcResultHandlers.print());

} catch (Exception e) {

e.printStackTrace();

}

/\*

// MediaType.APPLICATION\_JSON\_UTF8

// Content-Type:"application/json;charset=UTF-8"

String json="{\"loginName\":\"root\"}";

try {

mvc.perform(MockMvcRequestBuilders.post("/user")

.contentType(MediaType.APPLICATION\_JSON\_UTF8) // Content-Type:"application/json;charset=UTF-8"

.accept(MediaType.APPLICATION\_JSON\_UTF8) // Accept:"application/json;charset=UTF-8"

.content(json)

).andExpect(MockMvcResultMatchers.status().isOk())

.andDo(MockMvcResultHandlers.print());

} catch (Exception e) {

e.printStackTrace();

}\*/

}

}

#### 2.5.3输出

MockHttpServletRequest:

HTTP Method = POST

Request URI = /user

Parameters = {loginName=[root]}

Headers = [Content-Type:"application/x-www-form-urlencoded", Accept:"application/json;charset=UTF-8"]

Body = <no character encoding set>

Session Attrs = {}

Handler:

Type = com.chen.springboottest.control.LoginController

Method = public com.chen.springboottest.entity.User com.chen.springboottest.control.LoginController.user(java.lang.String)

Async:

Async started = false

Async result = null

Resolved Exception:

Type = null

ModelAndView:

View name = null

View = null

Model = null

FlashMap:

Attributes = null

MockHttpServletResponse:

Status = 200

Error message = null

Headers = [Content-Type:"application/json;charset=UTF-8"]

Content type = application/json;charset=UTF-8

Body = {"loginName":"root","password":"123456"}

Forwarded URL = null

Redirected URL = null

Cookies = []

### 2.6文件上传

#### 配置上传路径和静态资源位置

**Application.yml：**

这些是自定义的内容，后续会讲解这些配置在java代码中如何使用

上传位置： d:/springtest/ ，

访问位置： localhost:8080/images/1.jpg 。

file:

uploadFolder: d:/springtest/

staticAccessPath: /images/\*\*

max-file-size: 10MB

max-request-size: 10MB

**Java代码**

[通过@Value("${file.staticAccessPath}")标签取用配置文件中的值](mailto:通过@Value(\"${file.staticAccessPath}\")标签取用配置文件中的值)

获取值重定义上传配置

**@Configuration**

**public class UploadFilePathConfig implements WebMvcConfigurer {**

**@Value("${file.staticAccessPath}")**

**private String staticAccessPath;**

**@Value("${file.uploadFolder}")**

**private String uploadFolder;**

**public void addResourceHandlers(ResourceHandlerRegistry registry) {**

**registry.addResourceHandler(staticAccessPath).addResourceLocations("file:"+uploadFolder);**

**}**

**}**

**@Configuration**

**public class UploadFileConfig {**

**@Value("${file.uploadFolder}")**

**private String uploadFolder;**

**@Value("${file.max-file-size}")**

**private String maxFileSize;**

**@Value("${file.max-request-size}")**

**private String maxRequestSize;**

**@SuppressWarnings("deprecation")**

**@Bean**

**MultipartConfigElement multipartConfigElement(){**

**MultipartConfigFactory factory = new MultipartConfigFactory();**

**factory.setLocation(uploadFolder);**

**factory.setMaxFileSize(maxFileSize);**

**// 单次请求最大上传文件大小**

**factory.setMaxRequestSize(maxRequestSize);**

**return factory.createMultipartConfig();**

**}**

**}**

### 2.7 整合Lombok

未能在项目中使用，在整合lombok.jar到eclipse中的时候，启动eclipse报错

Errors occurred during the build.

Errors running builder 'Java Builder' on project 'springboottest'.

java.lang.NullPointerException

原因未详。

#### 2.7.1实现原理

javac就支持“JSR 269 Pluggable Annotation Processing API”规范，只要程序实现了该API，就能在javac运行的时候得到调用

1. javac对源代码进行分析，生成一棵抽象语法树(AST)

2. javac编译过程中调用实现了JSR 269的Lombok程序

3. 此时Lombok就对第一步骤得到的AST进行处理，找到Lombok注解所在类对应的语法树       (AST)，然后修改该语法树(AST)，增加Lombok注解定义的相应树节点

4. javac使用修改后的抽象语法树(AST)生成字节码文件

#### 2.7.2 POJO类常用注解

**@Getter/@Setter:**可以作用类和方法上，生成对应的getter/setter方法。可以设定访问权限和是否懒加载

@Getter

@Getter(value = AccessLevel.PUBLIC) // 设置访问权限

@Getter(lazy = true) // 懒加载

**@ToString：**作用于类，覆盖默认的toString()方法，可以通过of属性限定显示某些字段，通过exclude属性排除某些字段。

**@EqualsAndHashCode：**作用于类，覆盖默认的equals和hashCode

**@NonNull：**主要作用于成员变量和参数中，标识不能为空，否则抛出空指针异常。

**@NoArgsConstructor：**生成无参构造器；

**@RequiredArgsConstructor：**生成包含final和@NonNull注解的成员变量的构造器；

**@AllArgsConstructor：**生成全参构造器

**@Data：**作用于类上，是以下注解的集合：@ToString @EqualsAndHashCode @Getter @Setter @RequiredArgsConstructor

**@Builder：**作用于类上，将类转变为建造者模式

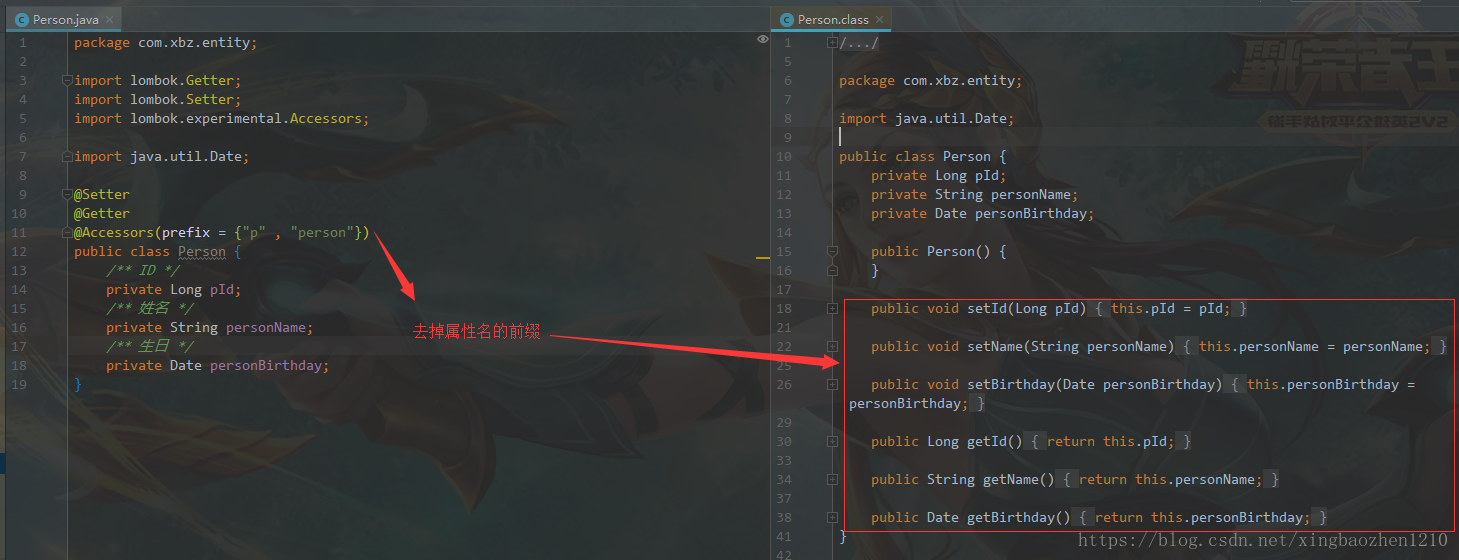
**@Log：**作用于类上，生成日志变量。针对不同的日志实现产品，有不同的注解：

**@Cleanup：**自动关闭资源，针对实现了java.io.Closeable接口的对象有效，如： 典型的IO流对象

**@SneakyThrows：**可以对受检异常进行捕捉并抛出，可以改写上述的main方法

**@Accessors**

prefix : 省略属性名中的前缀 , 例如pId属性 , 默认生成getPId() , 设置prefix = "p" , 则生成大的为getId() , 适用于属性名中包含一些前缀的类



**@chain：**

如果设置为true , 生成的set方法将返回this而不是void , 通过这个配置我们可以像jQuery一样进行链式编程

#### 2.7.3

#### 2.7.4

### 2.8 Spring Data JPA

#### 2.8.1 核心数据访问接口

**Repository**<T, ID extends Serializable>

**CrudRepository**<T, ID extends Serializable>

**PagingAndSortingRepository**<T, ID extends Serializable>

**JpaRepository**<T, ID extends Serializable>

四个类从上到下功能逐渐增强，实现接口就可以实现crud操作，可以简化数据访问的关键

**CrudRepository** 提供最基本的增删查改的操作

<S extends T> S save(S entity);

<S extends T> Iterable<S> save(Iterable<S> entities);

T findOne(ID id)

Boolean exists(ID id) // 判断该id的实体类是否存在

Iterable<T> findAll();

Iterable<T> findAll(Iterable<ID> ids)

Long count();

Void delete(ID id)

Void delete(T entity)

Void delete(Iterable<? extends T> entities)

Void deleteAll()

#### 2.8.2访问数据库时的时区（time zone）异常

The server time zone value '�й���׼ʱ��' is unrecognized or represents more than one time zone. You must configure either the server or JDBC driver (via the serverTimezone configuration property) to use a more specifc time zone value if you want to utilize time zone support.

**解决办法;**

在连接字符串后面加上?serverTimezone=UTC

其中UTC是统一标准世界时间。

完整的连接字符串示例：jdbc:mysql://localhost:3306/test?serverTimezone=UTC

或者还有另一种选择：

jdbc:mysql://127.0.0.1:3306/test?useUnicode=true&characterEncoding=UTF-8，这个是解决中文乱码输入问题，当然也可以和上面的一起结合：

jdbc:mysql://127.0.0.1:3306/test?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC

### 2.8.3

### 2.9统一异常处理

处理一般404和400页面。如有则返回对应的404.jsp 等其他页面

import javax.servlet.http.HttpServletRequest;

import org.apache.log4j.Logger;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.web.servlet.error.ErrorController;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import com.alibaba.fastjson.JSONObject;

import com.chen.springboottest.entity.ErrorResult;

@Controller

public class CodeExceptionAdvice implements ErrorController {

private final Logger log = Logger.getLogger(CodeExceptionAdvice.class);

@Autowired

HttpServletRequest request;

@RequestMapping("/error")

public String getErrorPath() {

Integer statusCode =

(Integer) request.getAttribute("javax.servlet.error.status\_code");

String url = (String) request.getAttribute("javax.servlet.error.request\_uri");

log.info("异常：" + statusCode);

result.setErrorCode(String.valueOf(statusCode));

result.setMsg(e.getMessage()+"["+e.getClass()+"]");

request.setAttribute("uri", url);

request.setAttribute("msg", e.getMessage());

request.setAttribute("code", String.valueOf(statusCode));

switch (statusCode) {

case 404:

return "/404";

case 400:

return "/400";

case 403:

return "/403";

default:

return "/500";

}

}

}

在servlet中对异常的配置，需在web.xml中配置如下：

*ErrorHandler* 的 Servlet 在任何已定义的异常或错误出现时被调用

<!-- servlet 定义 -->

<servlet>

<servlet-name>ErrorHandler</servlet-name>

<servlet-class>ErrorHandler</servlet-class>

</servlet>

<!-- servlet 映射 -->

<servlet-mapping>

<servlet-name>ErrorHandler</servlet-name>

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

</servlet-mapping>

<!-- error-code 相关的错误页面 -->

<error-page>

<error-code>404</error-code>

<location>/ErrorHandler</location>

</error-page>

<error-page>

<error-code>403</error-code>

<location>/ErrorHandler</location>

</error-page>

<!-- exception-type 相关的错误页面 -->

<error-page>

<exception-type>

javax.servlet.ServletException

</exception-type >

<location>/ErrorHandler</location>

</error-page>

<error-page>

<exception-type>java.io.IOException</exception-type >

<location>/ErrorHandler</location>

</error-page>

所有的异常有一个通用的错误处理程序

<error-page>

<exception-type>java.lang.Throwable</exception-type >

<location>/ErrorHandler</location>

</error-page>

以下表格属使用如request.getAttribute(“javax.servlet.error.status\_code”)，具体如何使用可以看上文的异常统一处理代码

|  |  |
| --- | --- |
| **序号** | **属性 & 描述** |
| 1 | **javax.servlet.error.status\_code** 该属性给出状态码，状态码可被存储，并在存储为 java.lang.Integer 数据类型后可被分析。 |
| 2 | **javax.servlet.error.exception\_type** 该属性给出异常类型的信息，异常类型可被存储，并在存储为 java.lang.Class 数据类型后可被分析。 |
| 3 | **javax.servlet.error.message** 该属性给出确切错误消息的信息，信息可被存储，并在存储为 java.lang.String 数据类型后可被分析。 |
| 4 | **javax.servlet.error.request\_uri** 该属性给出有关 URL 调用 Servlet 的信息，信息可被存储，并在存储为 java.lang.String 数据类型后可被分析。 |
| 5 | **javax.servlet.error.exception** 该属性给出异常产生的信息，信息可被存储，并在存储为 java.lang.Throwable 数据类型后可被分析。 |
| 6 | **javax.servlet.error.servlet\_name** 该属性给出 Servlet 的名称，名称可被存储，并在存储为 java.lang.String 数据类型后可被分析。 |

### 2.10

## MyBlog

### 3.1如何防止sql注入安全问题

1. 根据Mapper文件寻找匹配检测口
2. 测试会有问题的地方

#### 3.1.1根据Mapper文件寻找匹配检测口

* 修改、插入会在paraseMapByEntity方法中操作属性这或许是个检测的地方
* 还有一个地方就是Page、QueryHelper paramBind的时候
* 在没通过paraseMapByEntity方法的情况下：
* 在control请求方法体内对实体对象进行注入检测 // 改动多
* 在control请求方法中的参数进行注解标注，并对实体对象需要注入检测 // 改动多

#### 3.1.2可能存在隐患的串

sql 可以插入完整的任何sql语句 例如 ‘SELECT \* FROM `admin`’

columnSql 查询列名，插入列名 例如 ‘name,age’ 或者 ‘\*’

whereSql 添加查询语句 只需要列出查询条件即可，例如 ‘ and id = 1234565’

likeSql 模糊查询 例如 ‘name like %123%’

orderBySql 排序 例如 ·ORDER BY `create\_time` DESC·

limitSql 限制查询数量 例如 ‘LIMIT 0, 1000’

updateSql 修改语句 例如 ‘name = 'cjw', age = 5’

values 插入值 例如 ‘'123456','cjw',12’

**排除问题**

table 表名 例如 ‘user,admin’

#### 3.1.3解决

使用aop切面进行前置参数检测，但是aop不生效，暂时无法解决问题

#### 3.1.3xxx

## Vue

## Bat

lib目录所在D:\JAVA\dev\bat

已经在环境变量中创建了一个全局的lib\_path量，用于所有脚本运行的目录依据

并且可以在任何地方调用命令，因为已经在path中添加了环境路径

### 5.1使用封装命令：

String substring 中华人民共和国成立 2 5 // 在字符串中截取2到5

String endsWith test.java .java // 判断结尾串 是为1 、否为0

String length 中华人民共和国成立 // 返回串长度

String indexOf 中华人民共和国成立 华 // 查询某字符串在总串的位置

String lastIndexOf 中华人民共和国成立 成 //

String reverse 123456 // 结果是 654321 生成反串

String replace 121212 2 1 // 结果是 111212 ；三个参数

String replaceAll 121212 2 1 // 结果是 111111 ；三个参数

String 中华人民共和国成立 //

String 中华人民共和国成立 //

System print // 输出data/temp.data 中的结果值

### 5.2必要语句

#### 5.2.1 If语句

if %len% EQU 1 set preStr=%endsWithStr:~-1%

的

if "%compStr%" EQU "" (

echo 0 > %data\_path%temp.data

goto endd

)

以上是判断compStr是否等于空串

#### 5.2.2循环语句

##### for

for /f "delims=" %%i in ('dir /s /b /a-d') do (

if %%~zi lss 9216 if %%~zi gtr 1024 (

        del /f /q "%%i"

    )

if %%~zi gtr 20480 (

        del /f /q "%%i"

    )

)

以上是处理大于1k小于9k删除，大于20k的删除的批量处理程序

##### xxx

#### 5.2.3 xxx

#### 5.2.4 xxx

### 5.3逻辑符

EQU - 等于

NEQ - 不等于

LSS - 小于

LEQ - 小于或等于

GTR - 大于

GEQ - 大于或等于

### 5.4命令

移动文件

Move /y 1.txt d:\123\

Move 1.txt d:\123\

复制文件

Copy 1.txt d:\123\

### 5.5单符号的作用

* 1. ~
     1. 在for中表示使用增强 的变量扩展
     2. 在%var：~n,m%中表示使用扩展环境变量指定位置的字符串
     3. 在set /a 中表示一元运算符，将操作数按位取反
  2. ！
     1. 在set /a 中一元运算符，表示逻辑非。比如 set /a a=！0 a为1
  3. @
     1. 隐藏命令本身的回显
  4. $
     1. 在findstr命令里面表示一行的结束
     2. 在prompt命令里面，表示将其后的字符转义
  5. %
     1. 在set /a中的二元运算符，表示算数取余
     2. 在for命令in之前表示遍历指标变量
     3. 批处理中，后接一个数表示引用本批处理当前执行时的指定的参数
  6. ^
     1. 取消特定字符的转义作用
     2. 在set /a 中的二元运算符，表示按位异或
     3. 在findstr /r 的[] 中表示不匹配指定的字符集
  7. &amp
     1. 命令连接字符。比如我要在一行文本上同时执行两个命令，就可以用&amp;命令连接这两个命令。
     2. 在set /a 中是按位与
  8. \*
     1. 代表任意个任意字符。即通配符
     2. 在set /a 中的二元运算符，表示算数乘法
     3. 在findstr /r 中表示将前一个字符多次匹配
  9. –
     1. 范围表示符。比如日期的查找，for命令里的tokens操作中就可以用到这个字符。
     2. 在findstr /r 中连接两个字符表示匹配范围
     3. 跟在某些命令的/后表示去反向的开关
     4. 在set /a
        1. 表示一个负数。
        2. 表示算术减运算。

### 案例：

#### 可输入并作为参数执行

set rootPath=

set /p "rootPath=输入运行的根目录（回车默认c:\Windows\System32）："

if defined rootPath (echo 使用默认路径) else set rootPath=c:\Windows\System32

#### 打开一个应用

cd C:\Program Files (x86)\Tencent\QQ\Bin

start "QQScLauncher" "QQScLauncher.exe"

#### 获取环境变量

reg获取环境变量.bat

set regpath=

HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Environment

set evname=JAVA\_HOME

set javapath="C:\Program Files\Java\jdk1.8.0\_73"

reg add "%regpath%" /v %evname% /d %javapath% /f

::reg add KeyName [/v EntryName|/ve（环境名称）] [/t DataType] [/s separator] [/d value（值）] [/f]

#### Xxx

#### Xxx

## 控制台开发

借鉴项目：E:\学习\java\testCode\工单管理系统的初始原形测试\program

1、页面框架/架构

2、表格格式输出

## Jar脱离环境打包成exe可执行程序

过往案例：

D:\workspace\backups\backup\test2

D:\test2

### 成功打包案例

成功案例E:\学习\java\项目加急\脱离环境运行程序\打包程序exe\例子

### 获取jar运行时依赖的class类过程

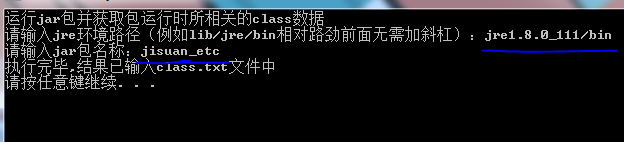
此处以操作jisuan\_etc为例





如图将JarClassData.bat 的任意目录，目录中需有目标文件jisuan\_etc.jar

然后运行cmd窗口



一次输入jre的bin路径和jar包名称

即可获得一个class.txt文件，里面就有运行依赖的类信息

在打包jar为可执行程序的时候可以根据使用JSmooth打包jar.docx的解说操作

# Vue

## 初次使用

环境准备并创建项目，此处记录从上到下是执行的记录，然后要想正常的使用一些命令，就得在path环境中配置命令cmd所在的文件夹路径

如果没设置path环境路径，如果没在cmd所在目录运行会提示命令不是内部或外部命令

先从nodejs.org中下载nodejs

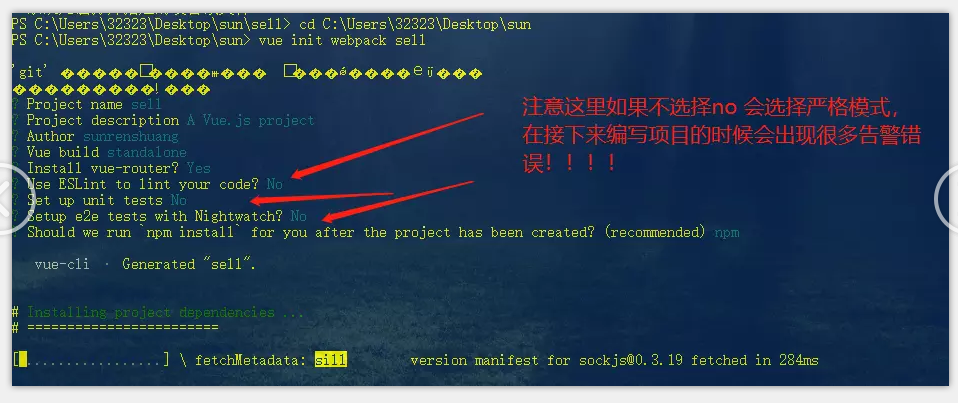
node -v 、 npm -v 检查版本是否是最新版本

如不是则通过这命令更新npm  install  -g  npm 到最新版本

npm install vue-cli -g //安装 vue-cli

vue list 查看是否成功安装

vue init webpack ”项目名称“



npm install -g cnpm --registry=https://registry.npm.taobao.org 使用淘宝NPM 镜像

cnpm -v 检查安装

npm run dev 报错：missing script:dev

vue init webpack my-project // 创建项目 建议先调用这个，系统会提示cli-init

npm install -g @vue/cli-init // 初始化

npm run dev 最后跳转到项目根目录然后调用此代码启动

注意：创建运行项目的时候要cd到项目的根目录，然后使用npm run dev命令

## 2.

## 3.初次使用

# HTML

## 助记

1. css动画案例收藏在html中的css文件中
2. 实用的页面挪到这了D:\gitcjw\liu\表白\font
3. 颜色

天蓝色：#3ea2ee

黑色： #888 、 #555

边框色（淡灰）： #dee2e6

1. 字体

font-family: -apple-system,BlinkMacSystemFont,"Segoe UI",Roboto,"Helvetica Neue",Arial,"Noto Sans",sans-serif,"Apple Color Emoji","Segoe UI Emoji","Segoe UI Symbol","Noto Color Emoji";

## 关于居中的问题

#### 案例一

使用padding使内部居中



这里用到的结构如下

<div class=”login”>

<div class=”login\_”>

</div>

</div>

login负责外面的定位 以及自身的的宽度 而高度由内部决定，高度很灵活

负责内部的内边距 不设宽度高度，依然是由内部元素决定

整个设计下来 可以达到居中，并且高度自适应

样式代码：

.login {width: 340px; background: #fff; position: relative; margin: auto; margin-top: 208px; }

.login\_ {padding: 50px;}



## 元素事件的问题

#### 案例一

Html:

<form id=” login\_form”>

<input type="submit" id="submit" class="input-bt-cs2" value="登录"/>

</form>

Js:

$("#login\_form").onsubmit = function (){

console.log("click");

//返回值为false 那么就不会进行下一步触发

//以当前为例 点击后下一步触发submit()并按照action跳转

//而如果为返回false 那么就只是点击 没有触发submit

//这里可以用来检查输入是否为空，验证码是否正确

return false;

}

## 关于浮动

#### 案例一

有些时候使用margin-top会让上一层的窗口也跟着margin 这给设计页面带来了些许小问题，那么此时上层窗口要使用清除浮动 并且position: relative; 来消除

# JSP

## Spring标签

结合Spring标签的使用：

### 1.1 <c:forEach></c:forEach>

begin= " " 从那个元素开始遍历，默认从0开始  
end=" " 到那个数据结束，默认到最后一个结束  
step=“ ” 步长 默认为1  
items=“ ” 需要遍历的数据集合  
var=“ ” 每个元素的名称  
varStatus= “ ” 当前正在遍历元素的状态对象（count属性：当前位置，从1开始）--序列

list集合：

<%

List<Student> list= new ArrayList<Student>();

list.add(new Student("张三","21"));

list.add(new Student("小明","32"));

list.add(new Student("小花","44"));

pageContext.setAttribute("list",list);%>

<c:forEach begin="0" end="2" items="${list}" step="1" var="Student" varStatus="varsta">

序号：${varsta.count} 姓名: ${Student.name} 年龄：${Student.age}<br></c:forEach>

map集合：

<%

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

map.put("011",new Student("张飞","44"));

map.put("012",new Student("小花","47"));

map.put("013",new Student("小明","52"));

pageContext.setAttribute("map",map);%><c:forEach begin="0" end="2" items="${map}" step="1" var="Student" varStatus="varsta">

序号：${varsta.count} 编号${Student.key} 姓名: ${Student.value.name} 年龄：${Student.value.age}<br></c:forEach>

### 1.2 <c:if test=""></c:if>

<c:if test="true">

判断输出

</c:if>

<c:if test="${30>22}">

判断输出

</c:if>

### 1.3 <jsp:include/>

包含标签

<jsp:include page="head.jsp"></jsp:include>

上面是头部

### 1.4 <c:set></c:set>

默认将数值存入page域中

<c:set var="mahuan" value="huanhuan"></c:set>

${mahuan}

scope="域对象"（page,request,session,application）

用于设置存入的域

<c:set var="mahuan" value="huanhuan" scope="request"></c:set>

${requestScope.mahuan}

### 1.5 <c:out value=""></c:out>

从域中输出值

<c:set var="mahuan" value="huanhuan" scope="request"></c:set><c:out value="${mahuan}"></c:out>

### 1.6 <c:choose></c:choose>

<c:choose></c:choose>

<c:when test=""></c:when>

<c:otherwise></c:otherwise>

多条件判断

<c:set var="mahuan" value="78" ></c:set><c:choose>

<c:when test="${mahuan>90 && mahuan<=100}">

特等奖

</c:when>

<c:when test="${mahuan>80 && mahuan<=90}">

一等奖

</c:when>

<c:when test="${mahuan>70 && mahuan<=80}">

二等奖

</c:when>

<c:when test="${mahuan>60 && mahuan<=70}">

三等奖

</c:when>

<c:otherwise>

参赛奖

</c:otherwise></c:choose>

### 1.7 使用标签注意需要导入的包

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

# xxx

# 学习的方向

1.对象导论：如何用面向对象的思路来开发

2. 深入JVM：Java运行机制以及JVM原理

3. 面向对象的特征：封装、继承、抽象、多态

4. 数组和容器：数组长度不可变性，容器的线程安全问题

5. I/O：IO工作原理和应用

6. 并发编程：concurrent包

7. 网络编程：TCP/IP+BIO/NIO UDP/IP+BIO/NIO

8. JDBC框架和反射：JNDI、连接池、annotation等

9. 其他：正则表达式、字符串等

在这里我还想多提一点，关于Java SE学到什么深度才合适。其实对于后续学习Java EE或者是Android来说，Java SE你只要学会皮毛就足够应付了，当然这里的皮毛是要求你熟练Java语法结构和基本CRUD操作。

可是这样真的又够了么？好多工作了一两年的程序员肯定又会慢慢怀疑自己的基础是否足够扎实，于是他们又拿出了《Thinking In Java》重新通读一遍。这里要说的是，这本书并不适合初学者，根本不应该出现在初学者的Java学习路线当中，不知道为啥网上很多人把他推荐给初学者。基础是非常重要的，越到后面你越会觉得基础知识的重要性。而真正有用的还是对Java的深入理解。所以如果你正在担心你的Java基础够不够，层次到没到应该有的深度，那你应该先庆幸下，至少你还想着如何提高自己的水平。

Java EE很杂，初学者一学到这里就会懵逼了，这块知识体系太庞大了，要学的东西太多，不过不要害怕，其实Java EE方向说好听点儿的叫企业级应用开发，说难听点儿的其实就是做网站。当然并不局限于做网站。

对于Java EE方向的学习，重点在于能够熟练的运用各种框架，以及掌握各种框架构成的核心思想。至于学什么框架，用什么应用服务器，在学习的时候可以采用容易配置的，所有的同类服务都是类似的，精通了一个，相似的学习就很快了。

下面来具体的看各个模块应该学点儿什么：

1.序列化：XML、JSON与Java之间的序列化和反序列化

2.ORM：推荐学习Hibernate

3.JSP/Servlet：可以顺带着把JS学习了

4. Spring（目前是当红炸子鸡）：IOC、AOP、Spring MVC

5. MVC：推荐学习Struts

6. Web Service：JDK自带web service、CXF等框架

7. SOA：了解ESB、MQ等企业级应用架构。

8. EJB：主要了解思路。可顺带着学习JBOSS

9. 其他：会使用JMS。了解JTA。

还是一点，Java EE方向的学习主要是会熟练运用即可。大点儿的企业都会有自己的框架，技术含量大同小异。会用即可。当然到了这个阶段设计模式的学习也必不可少，设计模式可以穿插于各个框架中进行学习

对于大多数企业来说，如果你能掌握上面提到的相关技术，并且熟悉SQL和Linux，那找个Java相关工作应该是没有问题的了。如果你想更深入的去应用上面所学的知识，分布式、搜索引擎、大数据处理等互联网热门技术都是你扩宽技能的方向。

开源的Lucene是搜索引擎技术最好的入门，简单易学，而且很多企业对搜索引擎的需求越来越大。

诸如MINA、Kafka这样的开源框架，是你步入分布式方向很好的基石。

如果你能了解HBase、Mongo这样的非关系型数据库，那就更有利于你思考大数据的问题。

技术是学不完的，而且更新的速度一定比你学习的速度还要快，不过技术基本上都是相通的，掌握其核心思想之后，再去学习其他的内容会很轻松，所以不用什么都学，把一些常用的学通就ok了。

下面这张图能够让你更好的理解哪些知识点重要。

