# 问题

## 1.jar包和war包的区别

1. war是一个web模块，其中需要包括WEB-INF，是可以直接运行的WEB模块；jar一般只是包括一些class文件，在声明了Main\_class之后是可以用java命令运行的。

2. war包是做好一个web应用后，通常是网站，打成包部署到容器中；jar包通常是开发时要引用通用类，打成包便于存放管理。

3. war是Sun提出的一种Web应用程序格式，也是许多文件的一个压缩包。这个包中的文件按一定目录结构来组织；classes目录下则包含编译好的Servlet类和Jsp或Servlet所依赖的其它类（如JavaBean）可以打包成jar放到WEB-INF下的lib目录下。

4. JAR文件格式以流行的ZIP文件格式为基础。与ZIP文件不同的是，JAR 文件不仅用于压缩和发布，而且还用于部署和封装库、组件和插件程序，并可被像编译器和 JVM 这样的工具直接使用。

# 基础

## 1.基本类型的互相转换

(1)String-->int

String str = “123456”;

int num = Integer.parseInt(str);

(2)int-->String

int num = 123;

String str = Integer.toString(num);

或

String str = “”+num;

(3)String-->long

String str = “123456”;

int num = Long.parseLong(str);

## 2.获取增强for的index

String[] data = new String[]{"rensanning001", "rensanning002", "rensanning003"};  
List<String> dataList = Arrays.asList(data);//**使用Arrays.asList()转化的list只能用来遍历**

int i = 0;  
for (String elem : dataList) {  
 System.out.println(i + ":" + elem);  
 i++;  
}

## 3.判断数据的类型

关键字 instanceof  用来判断数据的类型

boolean result = obj instanceof Class。

其中 obj 为一个对象，Class 表示一个类或者一个接口，当 obj 为 Class 的对象，或者是其直接或间接子类，或者是其接口的实现类，结果result 都返回 true，否则返回false。

例如：

String a = “哈哈”;

a instanceof String //true

## 4.包装类型的强制转换

(String)null 的结果依然是null(非字符串)

所以用String.valueOf()方法将null转转为”null”，再去判断”null”的方式可以换了。

(Integer)null 的结果依然是null(非字符串)

包装类型的强制转换应该是不影响null的，也不会报错。

## 5.对象初始化

Map<String,Object> map = new HashMap(12){{  
 put("1","a");  
}};  
  
List<String> list = new ArrayList(10){{  
 add("哈哈");  
}};  
  
SysUser sysUser = new SysUser(){{  
 setAge(12);  
 setName("姓名");  
}};

## 6.

# String

## 1.去除字符串最后一个字符

String code = “abcd/”;  
code = code.substring(0,code.length()-1);

## 2.去除字符串第一个字符

String code = “abcd/”;  
code = code.substring(1);

## 3.比较字符串是否相同

String s1 = “abc”;

String s2 = “cde”;

if(s1.equals(s2)){//如果相同

s1 = “fgh”;

}

## 4.去掉字符串两端空白符

String s1 = “ abc ”;

s1.trim();

## 5.字符串配合正则表达式过滤字符

String s1 = “，12的abc%……\*&，13a”;

s1.replaceAll(“，”,”,”); //将中文逗号替换为英文逗号

s1. replaceAll(“[^0-9\\,]”,””);//只保留数字和英文逗号，注意两个\\在代码里代表一个\

s1. replaceAll(“[^0-9]”,””);//只保留数字，注意两个[\\在代码里代表一个\](file:///\\在代码里代表一个\)

## 6.字符串的反转

String numberString = “abcde”;

numberString = new StringBuilder(numberString).reverse().toString();

## 7.字符串转日期

String startTime = "2020-10-20";

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm");  
startTime += " 00:00";

Date stTime = sdf.parse(startTime);//解析

## 8.split()的使用

可以将字符串分割为字符数组

(1) 单个符号作为分隔符。例：

String address="上海\上海市|闵行区\吴中路";

String[] splitAddress = address.split("\\"); //用\来分割字符串

String[] splitAddress = address.split("\\|"); //用|来分割字符串

(2) 多个符号作为分隔符。例：

String address="上海^上海市@闵行区#吴中路";

String[] splitAddress=address.split("\\^|@|#"); //用^、@、#来分割字符串

## 9.replace()和replaceAll()

1.replace()支持字符和字符串的全替换。

2. replaceAll()则支持规则表达式以及字符、字符串的全替换，它包含了replace()的功能。

3.replaceFirst()与replaceAll()的区别是仅支持第一个匹配的元素的替换。

4. val = val.replace(",", "','");

该语句可以将 张三,李四 变成 张三','李四 用于 sql的in语句来使用。

## 10.format()

String a = “1”;

String b = “2”;

String c = String.format(“update user set age = ‘%s’ where name = ‘%s’”,a,b);



## 11.随机生成指定位数的数字字符串

import org.apache.commons.lang.RandomStringUtils;

//获取一个随机的六位数字字符串  
String randomNumber = RandomStringUtils.*randomNumeric*(6);

## 12.

# StringJoiner

构造方法：

[StringJoiner](mk:@MSITStore:E:\个人资料\技术文档\jdk1.8中文版.CHM::/java/util/StringJoiner.html#StringJoiner-java.lang.CharSequence-)([CharSequence](mk:@MSITStore:E:\个人资料\技术文档\jdk1.8中文版.CHM::/java/lang/CharSequence.html) delimiter)：通过间隔符delimiter创建一个类似于字符串的

[StringJoiner](mk:@MSITStore:E:\个人资料\技术文档\jdk1.8中文版.CHM::/java/util/StringJoiner.html#StringJoiner-java.lang.CharSequence-)对象。

[StringJoiner](mk:@MSITStore:E:\个人资料\技术文档\jdk1.8中文版.CHM::/java/util/StringJoiner.html#StringJoiner-java.lang.CharSequence-java.lang.CharSequence-java.lang.CharSequence-)([CharSequence](mk:@MSITStore:E:\个人资料\技术文档\jdk1.8中文版.CHM::/java/lang/CharSequence.html) delimiter, [CharSequence](mk:@MSITStore:E:\个人资料\技术文档\jdk1.8中文版.CHM::/java/lang/CharSequence.html) prefix, [CharSequence](mk:@MSITStore:E:\个人资料\技术文档\jdk1.8中文版.CHM::/java/lang/CharSequence.html) suffix)：通过前缀

prefix，后缀suffix和间隔符delimiter创建一个类似于字符串的[StringJoiner](mk:@MSITStore:E:\个人资料\技术文档\jdk1.8中文版.CHM::/java/util/StringJoiner.html#StringJoiner-java.lang.CharSequence-)对象。

方法：

add([CharSequence](mk:@MSITStore:E:\个人资料\技术文档\jdk1.8中文版.CHM::/java/lang/CharSequence.html) newElement)：添加元素。

String toString()：返回String对象。

int length()：返回其长度(即字符个数)。

案例：

StringJoiner sj = new StringJoiner(":", "[", "]");

sj.add("George").add("Sally").add("Fred");

String desiredString = sj.toString();

结果：[George:Sally:Fred]

# StringBuilder

## 1.去掉最后一个字符

[StringBuilder](https://wenwen.sogou.com/s/?w=StringBuilder&ch=ww.xqy.chain) s = new StringBuilder();

s.append("zhidao");

s.deleteCharAt(s.length()-1);

## 2.清空内容

[StringBuilder](https://wenwen.sogou.com/s/?w=StringBuilder&ch=ww.xqy.chain) s = new StringBuilder();

s.delete(0,s.length());

或者

s.setLength(0);

# 日期

## 1.获取上个月的年份和月份

LocalDate lastMonth = LocalDate.*now*().minusMonths(1);  
String year = String.*valueOf*(lastMonth.getYear());  
String month = String.*valueOf*(lastMonth.getMonthValue());

## 2.获取当前年份和月份

LocalDate now = LocalDate.*now*();  
String year = String.*valueOf*(now.getYear());  
String month = String.*valueOf*(now.getMonthValue());

## 3.月份带0的格式化

//将年月格式转换为：2023-09 这种，月份小于10时带前缀0的格式

String.*format*("%s-%02d","2023",Integer.*parseInt*("09"))

## 4.

# 数组操作

## 1.将数组转化为字符串

String []arr = {“a”,”b”,”c”,”d”};

StringBuffer sb = new StringBuffer();  
for(int i=0;i<arr.length;i++){  
 sb.append(arr[i]);  
}  
String code = sb.toString();

或者

String []arr = {“a”,”b”,”c”,”d”};

StringBuffer sb = new StringBuffer();

Arrays.stream(arr).forEach(item -> {

sb.append(item);

});

String code = sb.toString();

## 2.将字符串数组转化为整形数组

String []arrCode = {“0”,”3”,”2”,”1”};  
int []arrIntSeq = new int[arrCode.length];  
//将排序的字符串数组转化为整形数组  
for(int i = 0;i < arrCode.length; i++){  
 arrIntSeq[i] = Integer.parseInt(arrCode[i]);  
}

## 3.得到一个整形数组中最大的数字

int []arrIntSeq = {1,3,5,2,9};

int max = arrIntSeq[0];  
for(int i = 1;i < size;i++){  
 if(arrIntSeq[i] > max){  
 max = arrIntSeq[i];  
 }  
}

## 4.遍历两个字符串数组是否有相同的元素

String []arr = {“ab”,”bc”,”cd”};

String []arrCode = {“ab”,”cd”,”gt”};

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

for(int j = 0;j<arrCode.length;j++){

if(arr[i].equals(arrCode[j])){

对比之后的操作

}

}

}

## 5.数组的三种初始化方式

1. 静态创建

Object[] arr1 = {"a","b","c"};

1. 动态创建

Object[] arr3 = new Object[]{"a","b","c"};

1. 动态创建

Object[] arr2 = new Object[3];

arr2[0] = "a";

arr2[1] = "b";

arr2[2] = "c";

1. 动态创建

Object[] arr4 = null;

arr4 = new Object[3];

arr4[0] = "a";

arr4[1] = "b";

arr4[2] = "c";

# List操作

## 1.判断List是否为空

list == null 意味着没有为list分配内存，它压根不存在

list.isEmpty()或者list.size()==0用于判断list里是否有元素

所以判断list时一般这么写:

if(list != null && !list. isEmpty()){不为空的操作}

但是有很多的工具类封装了类似方法，比如：

apache的CollectionUtils的isEmpty()和isNotEmpty()可以替换list != null && !list. isEmpty()。

## 2.在List存储不重复的数据

(1)可以使用list的contains()判断list是否包含该数据，再选择添加与否，但效率低。例：

if (!list.contains("home" + i)) {

list.add("home" + i);

}

(2)可以先用set集合判断，再插入到list中。例：

public static List<String> list = new ArrayList<String>();

public static Set<String> set = new HashSet<>();

for (int i = 0; i < 100000; i++) {

if (set.add("home" + i)) {

list.add("home" + i);

}

}

## 3.遍历时的注意事项

使用增强for遍历list时，不能remove和add元素。

若想遍历时remove元素，需要使用迭代器遍历，如：

List<Map<String, Object>> list = singleCriticalFormMapper.findOwnFormInfo(patInHosId);  
Iterator<Map<String, Object>> iterator = list.listIterator();  
while(iterator.hasNext()){  
 Map<String, Object> map = iterator.next();  
 //显示患者写过的记录单的记录，不包括当前记录  
 if(formRecordId.equals(map.get("formRecordId"))){  
 iterator.remove();  
 }

}

## 4. ArrayList的扩容机制

ArrayList初始化时默认容量为10

扩容时机：元素>实际容量

扩容大小：原容量+原容量右移一位(jdk1.7和1.8)

右移一位 会 向下取整

2.5向下取整是2，-3.5向下取整是-4

比如：若指定原容量为15，则扩容一次大小为：15+15/2=15+7=22

## 5.ArrayList初始化时添加元素

需要导入的包：

import com.google.common.collect.Lists;

newArrayList方法的源码：

public static <E> ArrayList<E> newArrayList(E... elements) {  
 Preconditions.checkNotNull(elements);  
 int capacity = computeArrayListCapacity(elements.length);  
 ArrayList<E> list = new ArrayList(capacity);  
 Collections.addAll(list, elements);  
 return list;  
}

调用方式举例：

ArrayList<Boolean> booleans = Lists.*newArrayList*(true,false);

# Map操作

## 1.判断Map是否为空

map == null 意味着没有为map分配内存，它压根不存在

map.isEmpty()或者map.size()==0用于判断 map 里是否有元素

所以判断 map时一般这么写:

if(map != null && !map. isEmpty()){不为空的操作}

但是有很多的工具类封装了类似方法，比如：

apache的MapUtils的isEmpty()和isNotEmpty()可以替换map != null && !map. isEmpty()。

## 2. HashMap的扩容机制

HashMap初始化时默认容量为16

当HashMap中的键值对数量>实际容量\*0.75时，HashMap会进行扩容，容量变为原来的2

倍。

## 3.遍历HashMap

(1)

(2)

## 4.HashMap的get(null)

当HashMap使用get(null)时，返回null。

# Web

## 1.后端Date类型数据传输到前端

(1)首先是得有一个实体类中有Date类型的属性

(2)使用该注解修饰实体类即可。如：

@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss",timezone = "GMT+8")

其中，pattern代表转换的格式，它是可以改变的。

(3)当然，也可以用这个注解直接修饰属性。

(4)原理是将Date类型格式化为String类型。

## 2.后端Long类型数据传输到前端

(1)首先是得有一个实体类中有Long/long类型的属性

(2)使用该注解修饰属性即可。如：

@JsonSerialize(using = ToStringSerializer.class)

(3)原理是将long类型转为String类型。如果不转换，则因为前端JS的number类型的数字位数比long类型的短导致long类型的数字可能会丢失精度，典型的比如雪花算法生成的id，到前端就会丢失精度。

## 3.计算页码

通过总行数和一页展示的行数来判断总页数

//int类型的totalRows为总行数，int类型的pageLineNumber为一页展示的行数

Integer totalPages = (int) Math.ceil(((double) totalRows) / pageLineNumber);

//Math.ceil()为向上取整，如：

Math.ceil(12.2) //13

Math.ceil(12.0) //12

# Spring

## 1. Assert.notNull()

该方法判断实体类或实体类的某个属性是否为null，为null则抛出异常信息。

用法：

User user = null;

Assert.notNull(user,"实体类user为空");

User user1 = new User();

Assert.notNull(user1.getName(),"用户名字为空");

## 2.

# Mybatis

## 1.Mapper接口方法里的形参问题

如果该方法里的形参既有实体类对象，又有String等基本类型的数据，那么需要给每个形参都加上注解@Param()。如：

List<NursingOrderVO> findNursingOrder(@Param("nursingOrder")NursingOrder nursingOrder, @Param("startTime") Date startTime, @Param("endTime")Date endTime);

而在Mapper.xml文件中使用实体类对象时，得加上它注解里的value。如：

<select id="findNursingOrder" resultType="NursingOrderVO">  
 SELECT \* FROM nmr\_nursing\_order  
 <where>  
 <if test="nursingOrder.patInHosId != null">  
 AND pat\_in\_hos\_id = #{nursingOrder.patInHosId}  
 </if>  
 <if test="startTime != null and endTime != null">  
 AND scheduled\_execution\_time &gt;= #{startTime} AND scheduled\_execution\_time &lt;= #{endTime}  
 </if>  
 <if test="nursingOrder.executeFlag != null">  
 AND execute\_flag = #{nursingOrder.executeFlag}  
 </if>  
 AND his\_org\_id = #{nursingOrder.hisOrgId}  
 AND use\_flag = '1'  
 </where>  
</select>

注意：当涉及到时间范围的操作，即使用 > 和 < 时，需要转义，&gt;是>,而&lt;是<

## 2.xml文件中bigint和varchar的转换(PG数据库)

cast(#{orgId} as bigint) <!--varchar转bigint-->

cast(#{hospitalId} as varchar) <!--bigint转varchar-->

## 3.if标签中判断为’’

<if test="babyId == ''">  
 AND a.BABY\_ID is null  
</if>

注意：非String类型不得判断是不是’’

## 4.if标签中判断为‘0’(单个字符)

<if test="babyId == '0'.toString()">  
 AND a.BABY\_ID is null  
</if>

或者

<if test=’babyId == “0”’>  
 AND a.BABY\_ID is null  
</if>

## 5.Mybatis中的if、else组合

<choose>  
 <when test="babyId != null and babyId != ''">  
 AND a.BABY\_ID = #{babyId}  
 </when>  
 <otherwise>  
 AND a.BABY\_ID is null  
 </otherwise>  
</choose>

其中choose为一个整体，when是if，otherwise是else。

## 5.实体类与数据库字段的对应

有时候想把数据库返回的字段直接封装到实体类中

(1)书写resultMap

<resultMap type="com.msun.middle.business.nursing.server.entity.RecordSheet" id="RecordSheet">  
 <id property="recordSheetID" column="FORM\_ID"/>  
 <result property="id" column="ID"/>  
 <result property="formName" column="FORM\_NAME"/>  
 <result property="formHeader" column=" FORM\_HEADER"/>  
 <result property="recordSheetID" column="FORM\_ID"/>  
</resultMap>

补充：<resultMap>标签中的type为对应的实体类，id为标识名，供sql标签引用。

<result>标签中property的值为实体类的属性名，column为数据库的字段名；

(2)在sql标签中引用id

<select id="getRecordSheetHead" parameterType="java.lang.String" resultMap="RecordSheet">  
 SELECT  
 FORM\_NAME,  
 FORM\_HEADER  
 FROM NG\_FORM\_LIST  
 WHERE FORM\_ID = #{formId}  
</select>

补充：<select>标签中resultMap的值即为(1)中<resultMap>标签中id的值，表示引用。这样

返回的字段数据就能直接封装到实体类中。

## 6.查询数据库并返回List

数据库字段与实体类映射正常，但没有查到数据时，返回的List为Empty，即size=0，而不

是null。

## 7.trim标签

trim标签常用于拼接动态的SQL。格式如下：

<trim prefix="" suffix="" suffixOverrides="" prefixOverrides=""></trim>

prefix：表示在trim包裹的SQL语句前面添加的指定内容。

suffix：表示在trim包裹的SQL末尾添加指定内容。

prefixOverrides：表示去掉（覆盖）trim包裹的SQL的指定首部内容。

suffixOverrides：表示去掉（覆盖）trim包裹的SQL的指定尾部内容。

举例：

"SELECT \* FROM s\_u\_user u WHERE 1=1" +  
"<trim prefix='and(' prefixOverrides='or' suffix=')'>" +  
 "<if test='edu\_train != null'>" +  
 " or u.user\_info\_education\_check\_status = 0 " +  
 "</if>" +  
 "<if test='leader\_admin != null'>" +  
 " or u.user\_info\_cadre\_admin\_check\_status = 0 " +  
 "</if>" +  
 "<if test='leader\_supervisor != null'>" +  
 " or u.user\_info\_cadre\_supervision\_check\_status = 0 " +  
 "</if>" +  
 "<if test='archivist != null'>" +  
 " or u.user\_info\_record\_check\_status = 0 " +  
 "</if>" +  
"</trim>" +

当if标签中的四个字符串都是null时相当于：

SELECT \* FROM s\_u\_user u WHERE 1 = 1

当if标签中只有archivist不是null时相当于：

SELECT \* FROM s\_u\_user u WHERE 1 = 1 and ( u.user\_info\_record\_check\_status = 0 )

当if标签中只有leader\_supervisor、archivist不是null时相当于：

SELECT \* FROM s\_u\_user u WHERE 1 = 1

and ( u.user\_info\_cadre\_supervision\_check\_status = 0 or

u.user\_info\_record\_check\_status = 0 )

## 8.

# Mybatis-Plus

## 1.生成雪花id的方法

IdWorker.*getId*()

或者

IdWorker.*getId*(new Object())

说明：第二个方法的参数可以是任意对象。

## 2.QueryWrapper

QueryWrapper是Mybatis-Plus3.x及以上版本使用的查询条件构造器。

## 3.EntityWrapper

EntityWrapper是Mybatis-Plus2.x及以下版本使用的查询条件构造器。

## 4.自带的单表方法

(1)Service方法

boolean save(T entity)

boolean saveBatch(Collection<T> entityList)

boolean saveBatch(Collection<T> entityList, int batchSize)

boolean saveOrUpdate(T entity)

boolean saveOrUpdate(T entity, Wrapper<T> updateWrapper)

boolean saveOrUpdateBatch(Collection<T> entityList)

boolean saveOrUpdateBatch(Collection<T> entityList, int batchSize)

T getById(Serializable id)

T getOne(Wrapper<T> queryWrapper)

T getOne(Wrapper<T> queryWrapper, boolean throwEx)

public Map<String, Object> getMap(Wrapper<T> queryWrapper)

public <V> V getObj(Wrapper<T> queryWrapper, Function<? super Object, V> mapper)

boolean remove(Wrapper<T> queryWrapper)

boolean removeById(Serializable id)

boolean removeById(Serializable id, boolean useFill)

boolean removeById(T entity)

boolean removeByIds(Collection<?> list)

boolean removeByIds(Collection<?> list, boolean useFill)

boolean removeBatchByIds(Collection<?> list, int batchSize)

boolean removeBatchByIds(Collection<?> list, int batchSize, boolean useFill)

boolean removeBatchByIds(Collection<?> list)

boolean removeBatchByIds(Collection<?> list, boolean useFill)

boolean removeByMap(Map<String, Object> columnMap)

boolean update(Wrapper<T> updateWrapper)

boolean update(T entity, Wrapper<T> updateWrapper)

boolean updateById(T entity)

boolean updateBatchById(Collection<T> entityList)

boolean updateBatchById(Collection<T> entityList, int batchSize)

List<T> list()

List<T> list(Wrapper<T> queryWrapper)

List<T> listByIds(Collection<? extends Serializable> idList)

(2)Mapper方法

## 5.

# Jackson

说明：jackson是Spring自带的json解析工具。

## 1.将json形式的字符串转化为Map格式

比如字符串logtype：

ObjectMapper objectMapper = new ObjectMapper();  
map = objectMapper.readValue(logtype,HashMap.class);  
logtype = String.valueOf(map.get("logtype"));

补充：也可以将HashMap.class换成任意实体类的.class文件，返回值将是那个实体类。

## 2.将数组形式的json字符串转化为List<Map>

ObjectMapper mapper = new ObjectMapper();  
List<Map> fieldRelationList = mapper.readValue(fieldReltaionStr, new TypeReference<List<Map>>() {});

# Fastjson

## 1.json形式的字符串转成jsonObjcet

JSONObject jsonObject = JSONObject.*parseObject*(json);

## 2.往jsonObject里添加元素

jsonObject.put("organizationId",resultMap.get("organizationId"));

## 3.jsonObject转成json形式的字符串

jsonObject.toString();

# Hutool工具包

## 1.判断集合是否为空

CollUtil.isEmpty(Collection<?> collection)

CollUtil.isNotEmpty(Collection<?> collection)

## 2.判断String是否为空

StrUtil.isEmpty(CharSequence str)

StrUtil.isNotEmpty(CharSequence str)

## 3.根据索引删除数组中指定的元素

ArrayUtil.remove(continueFormsArr,i);

补充：continueFormsArr是数组，i是下标，该方法返回删除元素后的数组

## 4.json操作

(1)将String转化为JSONObject

JSONUtil.parseObj(formSetUp)

补充：返回 cn.hutool.json.JSONObject 对象

(2)根据JSONObject中的key获取对应的value

jsonObject.getStr("personalInformation")

补充：返回 String 对象

(3)将String转化为JSONArray

JSONUtil.parseArray(personalInfoValue)

补充：返回 cn.hutool.json.JSONArray 对象

(4)JSONArray通过索引获取JSONObject对象

jsonArray.getJSONObject(k)

补充：返回 cn.hutool.json.JSONObject 对象

jsonArray.remove(k);//根据索引删除元素

jsonArray.put(k,a);//根据索引添加元素

(5)判断JSONObject是否为null

JSONObject.isNull();//用需要判断的对象调用isNull()

## 5.判断过去的时间距现在多少天

DateUtil.between(birthDate,new Date(), DateUnit.DAY)

补充：birthDate为过去的时间，是Date对象。

该方法返回Long对象，即距离现在的天数。

## 6.String类型的日期转换为Date类型

Date date = Convert.toDate(“2016-10-22 12:20”);

补充：任何类型的日期格式都可以用toDate()转换。

## 7.生成雪花id的方法

Long snowflakeID = IdUtil.*getSnowflake*(0,0).nextId();

## 8.

# Lambda和Stream

## 1.遍历集合

//遍历ArrayList

List<String> list = new ArrayList<>(6);  
list.add("1");  
list.add("2");  
list.forEach(v -> System.*out*.println(v));

//遍历HashMap

Map<String,Object> map = new HashMap<>(16);  
for(int i = 0; i < 10; i++){  
 map.put("key:"+i,"value:"+i);  
}  
map.forEach((k,v) -> {  
 System.*out*.println(k + v);  
});

## 2.将List转换为Map

List<SysUser> userList = new ArrayList<>();  
SysUser user1 = new SysUser(1L,"hah");  
SysUser user2 = new SysUser(2L,"jiji");  
userList.add(user1);  
userList.add(user2);  
Map<Long, SysUser> collect = userList.stream().collect(Collectors.*toMap*(item -> item.getId(), item -> item));

说明：将List中SysUser的id属性当作key，将SysUser当作value构建了collect。

## 3.过滤被@TableField注解修饰，且其属性exist = false的字段

public void testFilterField() {  
 Field[] declaredFields = SysUser.class.getDeclaredFields();  
 declaredFields = Arrays.*stream*(declaredFields).filter(f -> !fieldIsDecorateByTableField(f)).collect(Collectors.*toList*()).toArray(new Field[]{});  
 System.*out*.println(declaredFields);  
}

public boolean fieldIsDecorateByTableField(Field field){  
 boolean result = false;  
 TableField annotation = field.getAnnotation(TableField.class);  
 if(annotation != null && annotation.exist() == false){  
 result = true;  
 }  
 return result;  
}

## 4.

# 反射

## 1.遍历一个实体类的属性及其值(不会取它父类的属性)

public void getValue(SelectIdListDto selectIdListDto){

Field[] field = selectIdListDto.getClass().getDeclaredFields();

for(int j=0 ; j<field.length ; j++){

String name = field[j].getName();

name = name.substring(0,1).toUpperCase()+name.substring(1);

String type = field[j].getGenericType().toString();

if(type.equals("class java.lang.String")){//只要String类型的字段

Method m;

String value;

try {

m = selectIdListDto.getClass().getMethod("get"+name);

value = (String) m.invoke(selectIdListDto);

if(value != null && !"".equals(value)){

System.out.println(name);

System.out.println(value);

}

} catch (IllegalAccessException e) {

e.printStackTrace();

} catch (IllegalArgumentException e) {

e.printStackTrace();

} catch (InvocationTargetException e) {

e.printStackTrace();

} catch (NoSuchMethodException e) {

e.printStackTrace();

} catch (SecurityException e) {

e.printStackTrace();

}

}

}

}

## 2.

# 阿里云发送国际短信

注意：短信模板的内容不能修改，要想改，只能重新申请个短信模板，再审核。

## 1.pom文件引入依赖

<!-- 阿里云短信 -->  
<dependency>  
 <groupId>com.aliyun</groupId>  
 <artifactId>aliyun-java-sdk-core</artifactId>  
 <version>4.5.17</version>  
</dependency>  
<dependency>  
 <groupId>com.aliyun</groupId>  
 <artifactId>aliyun-java-sdk-dysmsapi</artifactId>  
 <version>2.1.0</version>  
</dependency>

## 2.代码实现

public static ServiceResult sendSms(String phone, String codeType, Map params) {  
 // 获取短信配置  
 AliyunSmsConfig.Template template = *aliyunSmsConfig*.getTemplate();  
 String templateCode;  
 if (EnumUtils.SmsTemplate.*REGISTER*.getName().equalsIgnoreCase(codeType)) {  
 templateCode = template.getCodeRegister();  
 } else if (EnumUtils.SmsTemplate.*AUTH*.getName().equalsIgnoreCase(codeType)) {  
 templateCode = template.getCodeAuth();  
 } else if (EnumUtils.SmsTemplate.*LOGIN\_CONFIRM*.getName().equalsIgnoreCase(codeType)) {  
 templateCode = template.getCodeLoginConfirm();  
 } else if (EnumUtils.SmsTemplate.*LOGIN\_ERROR*.getName().equalsIgnoreCase(codeType)) {  
 templateCode = template.getCodeLoginError();  
 } else if (EnumUtils.SmsTemplate.*INFO\_CHANGE*.getName().equalsIgnoreCase(codeType)) {  
 templateCode = template.getCodeInfoChange();  
 } else if (EnumUtils.SmsTemplate.*CHANGE\_PASSWORD*.getName().equalsIgnoreCase(codeType)) {  
 templateCode = template.getCodeChangePassword();  
 } else if(EnumUtils.SmsTemplate.*CHANGE\_MOBILE*.getName().equalsIgnoreCase(codeType)){  
 templateCode = template.getCodeChangePhone();  
 } else if(EnumUtils.SmsTemplate.*CHANGE\_MOBILE\_EN*.getName().equalsIgnoreCase(codeType)){  
 templateCode = template.getCodeChangePhoneEn();  
 } else if(EnumUtils.SmsTemplate.*VERIFICATION\_CODE*.getName().equalsIgnoreCase(codeType)){  
 templateCode = template.getVerificationCode();  
 }else if(EnumUtils.SmsTemplate.*VERIFICATION\_CODE\_EN*.getName().equalsIgnoreCase(codeType)){  
 templateCode = template.getVerificationCodeEn();  
 }else if(EnumUtils.SmsTemplate.*CREATE\_NEW\_USER*.getName().equalsIgnoreCase(codeType)){  
 templateCode = template.getCreateNewUser();  
 }else if(EnumUtils.SmsTemplate.*CREATE\_NEW\_USER\_EN*.getName().equalsIgnoreCase(codeType)){  
 templateCode = template.getCreateNewUserEn();  
 }else {  
 CCLog.*wLog*(new ErrorLog("手机号【"+phone+"】发送短信的验证码类型{"+codeType+"}不存在",new Exception()));  
 return new ServiceResult(false,"500", "验证码类型不存在");  
 }  
  
 // 组装请求对象  
 SendSmsRequest request = new SendSmsRequest();  
 request.setPhoneNumbers(phone);  
 // 短信签名  
 request.setSignName(template.getSignName());  
 // 短信模板  
 request.setTemplateCode(templateCode);  
 //如果传递了模板中的变量  
 if(MapUtils.*isNotEmpty*(params)){  
 // 变量替换  
 request.setTemplateParam(JSON.*toJSONString*(params));  
 }  
 try {  
 IClientProfile profile = DefaultProfile.*getProfile*(*aliyunSmsConfig*.getRegionId(), *aliyunSmsConfig*.getAccessKeyId(), *aliyunSmsConfig*.getAccessKeySecret());  
 IAcsClient acsClient = new DefaultAcsClient(profile);  
 // 执行请求  
 SendSmsResponse response = acsClient.getAcsResponse(request);  
 String smsCode = response.getCode();  
 if (HttpStatus.*OK*.getReasonPhrase().equalsIgnoreCase(smsCode)) {  
 return new ServiceResult(true,"200", "短信发送成功");  
 }  
 // 错误信息  
 return new ServiceResult(false,"500", response.getMessage());  
 } catch (Exception e) {  
 CCLog.*wLog*(new ErrorLog("手机号【"+phone+"】发送短信类型{"+codeType+"}的短信失败",new Exception()));  
 return new ServiceResult(false,"500", "短信发送失败");  
 }  
 }

## 3.参数设置

*codeType*是变量值，通过匹配枚举里的值来获取*templateCode。*

*templateCode*是短信模板，也可以直接用方法的参数传递。

*aliyunSmsConfig*.getRegionId()是发送短信的服务器地址。

#各种regionId(表示阿里云发送短信的服务器地址)  
### 亚太地区  
# 国内(杭州)  
regionId=cn-hangzhou  
# 香港  
#regionId=cn-hongkong  
# 日本  
#regionId=ap-northeast-1  
# 新加坡  
#regionId=ap-southeast-1  
# 澳大利亚  
#regionId=ap-southeast-2  
# 马来西亚  
#regionId=ap-southeast-3  
# 印度尼西亚  
#regionId=ap-southeast-5  
### 欧洲与美洲地区  
# 美国(硅谷)  
#regionId=us-west-1  
# 美国(弗吉尼亚)  
#regionId=us-east-1  
# 英国  
#regionId=eu-west-1  
# 德国  
#regionId=eu-central-1  
### 中东与印度地区  
# 印度  
#regionId=ap-south-1  
# 阿联酋  
#regionId=me-east-1

aliyunSmsConfig.getAccessKeyId()是 AccessKeyId

aliyunSmsConfig.getAccessKeySecret()是 Secret

# 正则表达式

## 1.sql语句中，不允许出现把\*当作通配符使用的情况。

//禁用select \* from这样的语句，匹配忽略大小写，可以区分\*号是通配符还是乘号，不允许\*当作通配符使用  
String patternStrSelect = "\\bselect\\b(.\*\\.|)\\s\*\\\*.\*\\bfrom\\b";  
Pattern patternSelect = Pattern.*compile*(patternStrSelect,Pattern.*CASE\_INSENSITIVE*);  
Matcher matcherSelect = patternSelect.matcher(content);  
if(matcherSelect.find()){  
 sr.setSuccess(false);  
 sr.setRtnInfo(“不允许将\*当作通配符使用”);  
 return sr;  
}

## 2.禁用for和while死循环

//禁用for while死循环  
String content = page.getPageSource();  
String patternStrFor = "for\\s\*\\(.\*;\\s\*(true)?\\s\*;.\*\\)";  
Pattern patternFor = Pattern.*compile*(patternStrFor);  
Matcher matcherFor = patternFor.matcher(content);  
String patternStrWhile = "while\\s\*\\(true\\)";  
Pattern patternWhile = Pattern.*compile*(patternStrWhile);  
Matcher matcherWhile = patternWhile.matcher(content);  
if(matcherWhile.find()) {  
 sr.setSuccess(false);  
 sr.setRtnInfo(LocalUtils.*getMessage*(this.userInfo.getLanguage(),"label.code.error")+"for(int i=0;true;i++)，while(true)");  
 return sr;  
}  
if(matcherFor.find()) {  
 sr.setSuccess(false);  
 sr.setRtnInfo(LocalUtils.*getMessage*(this.userInfo.getLanguage(),"label.code.error")+"for(int i=0;true;i++)，while(true)");  
 return sr;  
}

## 3.sql中禁用delete语句

//禁用delete语句，匹配忽略大小写  
String patternStrDelete = "\\bdelete\\b\\s\*from\\b";  
Pattern patternDelete = Pattern.*compile*(patternStrDelete,Pattern.*CASE\_INSENSITIVE*);  
Matcher matcherDelete = patternDelete.matcher(content);  
if(matcherDelete.find()){  
 sr.setSuccess(false);  
 sr.setRtnInfo(LocalUtils.*getMessage*(this.userInfo.getLanguage(),"label.code.error")+"delete");  
 return sr;  
}

# 工作问题

## 1.并发上传图片

File file = new File("img00\_upload");

将上传的文件都放到此file中，因为此file的名称固定，故只会创建一个文件来存储传递来的文件流，会导致并发时图片重复上传的问题。场景是当一个图片放到此文件中，未上传时，另一个图片将此文件中的内容覆盖了，导致上传的图片重复了。

解决方式：

(1)将文件的名称置为唯一性(比如UUID之类的)，上传完在finally代码块中直接

file.delete()。

(2)加锁。