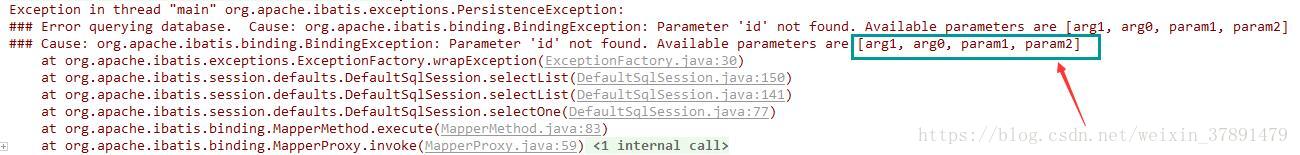
# 异常处理：

## mybatis中传入多个参数的4个解决方法

问题引入



想必大家可能遇到过类似的错误，当传入多个参数时，映射文件无法获得传入的参数

我的例子

// javabean

public class User {

private int id;

private String name;

private byte age;

// 忽略了getter、setter方法

}

// 接口

public interface UserMapper {

List<User> select(String name,byte age);

}

// 映射文件

<select id="select" resultType="model.User">

select \* from `user` where name = #{id} and age =#{age}

</select>

1. 利用参数出现的顺序

Parameter ‘id’ not found. Available parameters are [arg1, arg0, param1, param2]，这句话的意思就是id找不到，可用的参数是[arg1, arg0, param1, param2]。所以可使用参数出现的顺序号码引用参数，第一个参数用arg0或param1表示，第二个参数用arg1或param2表示，以此类推（arg从0开始计数，param从1开始计数）。

修改xml为如下方式：

1使用参数索引

<select id="select" resultType="model.User">

select \* from `user` where name = #{arg0} and age =#{arg1}

</select>

// or

<select id="select" resultType="model.User">

select \* from `user` where name = #{param1} and age =#{param2}

</select>

2.使用注解

给接口中方法的参数加上注解，xml不变

1

public interface UserMapper {

List<User> select(@Param("name") String name,@Param("age") byte age);

}

3.使用map

3.1把接口的形参改为Map

// 接口

public interface UserMapper {

List<User> select(Map params);

}

3.2 把需要的参数封装在map中

Map params = new HashMap();

params.put("name", "王尼玛");

params.put("age", 32);

List<User> list = userMapper.select(params);

4.把参数封装在Javabean中

4.1 把接口的形参改为javabean

// 接口

public interface UserMapper {

List<User> select(User user);

}

4.2 把需要的参数封装在javabean中

User user = new User();

user.setName("四");

user.setAge((byte)43);

List<User> list = userMapper.select(user);

so,我觉得应该把题目改为，mybatis中传入多个参数的2个解决方法。毕竟3、4是通过把多个参数转化为1个参数的方式解决问题的，哈哈。

5 拓展（接口中传入参数的各种情况）

5.1 当只有一个参数时

使用arg0,param1获得参数是一直有效的，但是个人不推荐这样做，毕竟看起来不优雅

5.1.1 参数是基本类型

xml中标签内的变量名可以随便写，#{id}，#{ids}，#{123}，#{xxgdsgdg},等都可以获得到参数。

5.1.2 参数是javabean

xml中标签内直接填写Javabean中的属性名

5.1.3 参数是数组

使用array获得参数，再用foreach循环

5.1.4 参数是List

使用list或collection获得参数，再用foreach循环

5.1.5 参数是Set

使用collection获得参数，再用foreach循环

5.2 当有多个参数时

使用argN、paramN或@param定位到某个参数，再灵活使用ognl就ok了。比如#{user.name}、#{users[0].name}。

## 前端异常

异常信息：

Uncaught DOMException: Failed to execute 'appendChild' on 'Node': The new child element contains the parent.

异常原因解决：HTML页面div标签异常

异常信息：Uncaught TypeError: Cannot read property 'substr' of undefined

原因：调用substr方法的字符串未定义undefined

异常信息：Uncaught ReferenceError: getLeftZeroStr is not defined

原因：getLeftZeroStr方法未定义

## XML映射文件异常

异常信息：org.mybatis.spring.MyBatisSystemException: nested exception is org.apache.ibatis.builder.BuilderException: An invalid property 'jdbcType ' was found in mapping #{xxx, jdbctype = VARCHAR}. Valid properties are javaType,jdbcType,mode,numericScale,resultMap,typeHandler,jdbcTypeName

异常原因解决：#{xxx, jdbctype = VARCHAR}中的jdbctype改为jdbcType

异常信息：Caused by: org.apache.ibatis.builder.BuilderException: Error resolving JdbcType. Cause: java.lang.IllegalArgumentException: No enum constant org.apache.ibatis.type.JdbcType.varchar

异常原因解决：#{xxx, jdbcType = varchar }. 中的varchar改为VARCHAR

## MySql使用异常

ERROR 1820 (HY000): You must reset your password using ALTER USER statement before executing this statement.

mysql> set password for 'root'@'host'=PASSWORD("123456");

查看mysql原始密码安装目录下的data文件夹下的\*.err文件里面找到

异常信息：

java.sql.SQLException: The server time zone value 'ÖÐ¹ú±ê×¼Ê±¼ä' 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.

at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:127)

at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:95)

at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:87)

at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:61)

at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:71)

at com.mysql.cj.jdbc.exceptions.SQLExceptionsMapping.translateException(SQLExceptionsMapping.java:76)

at com.mysql.cj.jdbc.ConnectionImpl.createNewIO(ConnectionImpl.java:862)

at com.mysql.cj.jdbc.ConnectionImpl.<init>(ConnectionImpl.java:444)

at com.mysql.cj.jdbc.ConnectionImpl.getInstance(ConnectionImpl.java:230)

at com.mysql.cj.jdbc.NonRegisteringDriver.connect(NonRegisteringDriver.java:226)

at com.alibaba.druid.pool.DruidAbstractDataSource.createPhysicalConnection(DruidAbstractDataSource.java:1558)

at com.alibaba.druid.pool.DruidAbstractDataSource.createPhysicalConnection(DruidAbstractDataSource.java:1623)

at com.alibaba.druid.pool.DruidDataSource$CreateConnectionThread.run(DruidDataSource.java:2468)

问题解决：

1、这是在使用MySQL 8.0以上版本（MySQL连接驱动和版本都是8.0以上）的时候出现的问题错误，我们需要在访问数据库的Url后面加上以下的语句即可：

?serverTimezone=GMT%2B8

2、设置数据库时区

show variables like '%time\_zone%';

set global time\_zone='+8:00';

MySQL 服务连接命令：net start mysql

用户登陆命令：mysql –u root –p

Enter password:password

net start mysql 发生系统错误 5。 拒绝访问。

用管理员身份重新打开命令行窗口即可

## node.js项目启动异常

1、npm run dev启动报如下异常：

Error: Cannot find module 'chalk'

at Function.Module.\_resolveFilename (internal/modules/cjs/loader.js:581:15)

at Function.Module.\_load (internal/modules/cjs/loader.js:507:25)

at Module.require (internal/modules/cjs/loader.js:637:17)

at require (internal/modules/cjs/helpers.js:22:18)

at Object.<anonymous> (E:\YongYing\Cloud-Platform\AG-Admin-v2.0\build\check-versions.js:1:75)

at Module.\_compile (internal/modules/cjs/loader.js:689:30)

at Object.Module.\_extensions..js (internal/modules/cjs/loader.js:700:10)

at Module.load (internal/modules/cjs/loader.js:599:32)

at tryModuleLoad (internal/modules/cjs/loader.js:538:12)

at Function.Module.\_load (internal/modules/cjs/loader.js:530:3)

npm ERR! code ELIFECYCLE

npm ERR! errno 1

npm ERR! AG-Admin@1.2.0 dev: `node build/dev-server.js`

npm ERR! Exit status 1

npm ERR!

npm ERR! Failed at the AG-Admin@1.2.0 dev script.

npm ERR! This is probably not a problem with npm. There is likely additional logging output above.

npm ERR! A complete log of this run can be found in:

npm ERR! C:\Users\lizhenzhong\AppData\Roaming\npm-cache\\_logs\2019-03-01T03\_26\_29\_783Z-debug.log

原因：未安装依赖，在项目目录下执行：npm install 命令即可

之后又有如下异常：

Error: Cannot find module 'escape-string-regexp'

at Function.Module.\_resolveFilename (internal/modules/cjs/loader.js:581:15)

at Function.Module.\_load (internal/modules/cjs/loader.js:507:25)

at Module.require (internal/modules/cjs/loader.js:637:17)

at require (internal/modules/cjs/helpers.js:22:18)

at Object.<anonymous> (E:\YongYing\Cloud-Platform\AG-Admin-v2.0\node\_modules\\_chalk@1.1.3@chalk\index.js:2:26)

at Module.\_compile (internal/modules/cjs/loader.js:689:30)

at Object.Module.\_extensions..js (internal/modules/cjs/loader.js:700:10)

at Module.load (internal/modules/cjs/loader.js:599:32)

at tryModuleLoad (internal/modules/cjs/loader.js:538:12)

at Function.Module.\_load (internal/modules/cjs/loader.js:530:3)

npm ERR! code ELIFECYCLE

npm ERR! errno 1

npm ERR! AG-Admin@1.2.0 dev: `node build/dev-server.js`

npm ERR! Exit status 1

npm ERR!

npm ERR! Failed at the AG-Admin@1.2.0 dev script.

npm ERR! This is probably not a problem with npm. There is likely additional logging output above.

npm ERR! A complete log of this run can be found in:

npm ERR! C:\Users\lizhenzhong\AppData\Roaming\npm-cache\\_logs\2019-03-01T03\_29\_23\_468Z-debug.log

原因：node\_modules中缺少模块文件，可以删除node\_modules文件后重新执行npm install命令，后再执行npm run dev

其中npm可以是cnpm

npm 报错This is probably not a problem with npm. There is likely additional logging output above.

可能由于种种版本更新的原因需要执行

npm install

重新安装一次，如果还是不可以的话，在把之前装的都清空

rm -rf node\_modules

rm package-lock.json

npm cache clear --force

npm install

## Invalid bound statement (not found):

检查 mapper.xml 中 namespace 等于是否 与 mapper.xml的文件名字一致

Unsatisfiehd dependency expressed through field 'service'

Service实现类，Mapperl类加@Service和@Mapper注解

## org.apache.ibatis.reflection.ReflectionException

Caused by: org.apache.ibatis.reflection.ReflectionException: There is no getter for property named 'userId' in 'class java.lang.Long'

at org.apache.ibatis.reflection.Reflector.getGetInvoker(Reflector.java:380)

at org.apache.ibatis.reflection.MetaClass.getGetInvoker(MetaClass.java:170)

at org.apache.ibatis.reflection.wrapper.BeanWrapper.getBeanProperty(BeanWrapper.java:152)

at org.apache.ibatis.reflection.wrapper.BeanWrapper.get(BeanWrapper.java:48)

at org.apache.ibatis.reflection.MetaObject.getValue(MetaObject.java:116)

at org.apache.ibatis.scripting.xmltags.DynamicContext$ContextMap.get(DynamicContext.java:97)

at org.apache.ibatis.scripting.xmltags.DynamicContext$ContextAccessor.getProperty(DynamicContext.java:116)

at org.apache.ibatis.ognl.OgnlRuntime.getProperty(OgnlRuntime.java:1657)

at org.apache.ibatis.ognl.ASTProperty.getValueBody(ASTProperty.java:92)

at org.apache.ibatis.ognl.SimpleNode.evaluateGetValueBody(SimpleNode.java:170)

at org.apache.ibatis.ognl.SimpleNode.getValue(SimpleNode.java:210)

at org.apache.ibatis.ognl.ASTNotEq.getValueBody(ASTNotEq.java:49)

at org.apache.ibatis.ognl.SimpleNode.evaluateGetValueBody(SimpleNode.java:170)

at org.apache.ibatis.ognl.SimpleNode.getValue(SimpleNode.java:210)

at org.apache.ibatis.ognl.Ognl.getValue(Ognl.java:333)

at org.apache.ibatis.ognl.Ognl.getValue(Ognl.java:413)

at org.apache.ibatis.ognl.Ognl.getValue(Ognl.java:395)

at org.apache.ibatis.scripting.xmltags.OgnlCache.getValue(OgnlCache.java:48)

at org.apache.ibatis.scripting.xmltags.ExpressionEvaluator.evaluateBoolean(ExpressionEvaluator.java:32)

at org.apache.ibatis.scripting.xmltags.IfSqlNode.apply(IfSqlNode.java:33)

at org.apache.ibatis.scripting.xmltags.MixedSqlNode.apply(MixedSqlNode.java:32)

at org.apache.ibatis.scripting.xmltags.DynamicSqlSource.getBoundSql(DynamicSqlSource.java:40)

at org.apache.ibatis.mapping.MappedStatement.getBoundSql(MappedStatement.java:278)

at org.apache.ibatis.executor.CachingExecutor.query(CachingExecutor.java:75)

at org.apache.ibatis.session.defaults.DefaultSqlSession.selectList(DefaultSqlSession.java:108)

at org.apache.ibatis.session.defaults.DefaultSqlSession.selectList(DefaultSqlSession.java:102)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

at java.lang.reflect.Method.invoke(Method.java:606)

at org.mybatis.spring.SqlSessionTemplate$SqlSessionInterceptor.invoke(SqlSessionTemplate.java:358)

... 89 more

出现原因

这里出现的问题是在DAO方法中定义的参数 与 实体中定义的属性不一致 导致的。

解决方案：

dao层加@Param("userId")注解即可

public List<DictItem> selectKeyByUserId(@Param("userId") long userId);

## org.apache.ibatis.binding.BindingException

mybatis配置时出现org.apache.ibatis.binding.BindingException: Invalid bound statement (not found)

如果出现：

org.apache.ibatis.binding.BindingException: Invalid bound statement (not found)

一般的原因是Mapper interface和xml文件的定义对应不上，需要检查包名，namespace，函数名称等能否对应上，需要比较细致的对比，我经常就是写错了一两个字母搞的很长时间找不到错误

按以下步骤一一执行：

1：检查xml文件所在的package名称是否和interface对应的package名称一一对应

2：检查xml文件的namespace是否和xml文件的package名称一一对应

3：检查函数名称能否对应上

4：去掉xml文件中的中文注释

5：随意在xml文件中加一个空格或者空行然后保存

一般来说到此就可以排除错误了

Cause: com.baomidou.mybatisplus.exceptions.MybatisPlusException: Failed to process, please exclude the tableName or statementId

原因：SQL语句字段和from之间多了个逗号，注意排查

报错内容:Error running 'ServiceStarter': Command line is too long. Shorten command line for ServiceStarter or also for Application default configuration.

解法:修改项目下 .idea\workspace.xml，找到标签 <component name="PropertiesComponent"> ， 在标签里加一行 <property name="dynamic.classpath" value="true" />

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## APPLICATION FAILED TO START

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Description:

Failed to configure a DataSource: 'url' attribute is not specified and no embedded datasource could be configured.

Reason: Failed to determine a suitable driver class

Action:

Consider the following:

If you want an embedded database (H2, HSQL or Derby), please put it on the classpath.

If you have database settings to be loaded from a particular profile you may need to activate it (no profiles are currently active).

Disconnected from the target VM, address: '127.0.0.1:53320', transport: 'socket'

Process finished with exit code 1

Maven clean之后要重启配置服务

## com.fasterxml.jackson.databind.exc.MismatchedInputException: Cannot deserialize instance of `java.lang.String` out of START\_ARRAY token

前端传回的是对象Array或者对象属性有Array类型，而@RequestBody 类型为对象或者对应对象属性不是Array导致异常，请仔细检查前后端对象及对象属性是否匹配

## Java.sql.SQLException: 数字溢出.............的异常

原因可能有以下两种，一：使用的jdbc驱动与数据库版本不匹配引起的，把版本改为一致的，测试看看；

二：可能是数据库的某个NUMBER类型的字段超出对应实体类属性的接收范围(如数据库字段是11位的手机号，而对应实体类使用Integer接收Phone这个属性，必然溢出)，

解决方法：将实体类该属性的数据类改为Long即可。

## JSON parse error: Cannot deserialize value of type `java.util.Date` from String"2019-09-11"

org.springframework.http.converter.HttpMessageNotReadableException: JSON parse error: Cannot deserialize value of type `java.util.Date` from String "2019-09-11": not a valid representation (error: Failed to parse Date value '2019-09-11': Unparseable date: "2019-09-11"); nested exception is com.fasterxml.jackson.databind.exc.InvalidFormatException: Cannot deserialize value of type `java.util.Date` from String "2019-09-11": not a valid representation (error: Failed to parse Date value '2019-09-11': Unparseable date: "2019-09-11")

对应实体类属性加注解

@JsonFormat(shape = JsonFormat.Shape.*STRING*, pattern = "yyyy-MM-dd", timezone = "GMT+8")

# 前端技巧

## button 按钮 的 disabled 两种状态 (启用和禁用)

禁用按钮：

$("#id").attr("disabled","true");

$("#id").attr("disabled",true);

$("#id").attr("disabled","disabled");

启用按钮：

$("#id").removeAttr("disabled");

$("#id").attr("disabled",false);

注意：

1、$("#id").attr("disabled","false");不起作用

2、disabled只能禁用button，对超链接不起作用

## Bootstrap-table 表格列属性调整：

隐藏列：visible:false eg：{field: 'union\_code', title:'联合代码',visible:false}

列排序按钮：sortable: true

列内容左对齐：align:'left'

列显示为输入框：

formatter:function(value, row, index) {

var input = '<input id="rating'+index+'" rows="1" style="width:80px"></input>'

return input;

}

输入框获取焦点：document.getElementById('inputid').focus();

## 设置单元格css样式

columns: [{

　　field: 'index',

　　title: '序号',

　　align:"center",

　　formatter:function(value, row, index){

　　　　return index+1;

　　}，

　　cellStyle:{

　　　　css:{"background-color":"red"}

　　}

}]

根据单元格或者同行内其他单元格内容设置该单元格的css样式

修改上面代码：

cellStyle:function(value,row,index){

　　　if (value==1){

　　　　　　return {css:{"background-color":"red"}}

　　　}else{

　　　　　　return {css:{"background-color":"green"}}

　　　}

}

根据内的内容，给该行设置一定的css样式

rowStyle:function(row,index){

　　if (index==1){

　　　　return {css:{"background-color":"red"}}

　　}else{

　　　　return {css:{"background-color":"green"}}

　　}

},

columns: [...]

或者：按需求设置不同的样式：5个取值代表5中颜色['active', 'success', 'info', 'warning', 'danger'];

rowStyle: function (row, index) {

var style = "";

style='danger';

return { classes: style }

}

或者：自定义字体颜色，或者背景颜色

rowStyle: function (row, index) {

var style = {};

//style={css:{'color':'#ed5565'}};

style={css:{'background-color':'#f0f0f0','color':'red'}};

return style;

}

设置单元格是否可用：

{field: 'checkbox', checkbox: true, formatter: function(value,row,index){

var style = {};

if (row.isWaitRating > 0) {

style = {disabled:true};

}

return style;

}}

## substr(start, [length])

截取从指定下标 start 开始，长度为 length 的字符串。

如果 start 是负数，则从字符串尾部开始计数，比如-1表示从倒数第一个开始。如果这个负数的绝对值大于字符串长度，那么会认为是 0。

如果length 是0或者负数，则会返回一个空字符串，如果省略 length，则会截取从start到结尾。

substring(start, [stop])

截取从下标start到stop（不包括stop位置的字符）的字符串。

如果省略stop，则截取到结尾。start和stop必须为非负整数，也就是 0，1，2 …

如果start大于stop，js会自动交换它们，然后计算。

如果start等于stop，则返回空字符串。

## Vue脚手架安装和卸载

全局安装：npm install vue-cli -g;

全局卸载：npm uninstall vue-cli -g;

查看vue版本，vue -V 回车，查看vue最新的版本。

运行环境：

node.js：https://nodejs.org/en/ 建议安装LTS版本

npm install -g cnpm --registry=https://registry.npm.taobao.org 执行该命令，npm国内速度很慢，建议使用阿里提供的镜像。

项目运行：

1、进入项目根目录。

2、执行 cnpm install 安装依赖

3、执行 cnpm run dev 启动项目

## Yarn知识初解

yarn 是什么

yarn 是新型的 javascript 包管理器，它是npm的替代品。它旨在解决npm解决不了的，在跨平台与跨用户之间安装依赖时的代码一致性问题、在安装依赖时花费太长时间、以及 npm 客户端自动执行某些依赖库的代码所导致的安全性问题。

所以相对于npm, yarn 具有快速，安全，可靠的特点。

(有同学表示，cnpm也不错，这个我承认，但是cnpm有个致命缺陷，用它下载安装的模块都是以软链形式存在的，本来我们的模块文件就多，再加个软链又多一倍文件，导致有些编辑器（sublime text）和 IDE（WebStorm）检索目录时非常慢，甚至卡死。作为一名webstorm使用爱好者，这一点是我不能忍受的。)

如何使用

第一步当然需要安装了，可以直接去官网下载安装，当然对于国内用户，更推荐去yarn官方中文网下载安装，它和英文官网是同步更新的。

安装完毕后，重点来了，我们下载的资源包默认地址都是在国外，由于众所周知的原因，往往会下的很慢，甚至下载失败。所以我们要配置yarn的下载依赖包的仓库地址，使用淘宝镜像是目前最好的选择:

yarn config set registry https://registry.npm.taobao.org -g

运行之后，我们可继续运行下面这条命令来查看，yarn的下载依赖包的仓库地址是否已经改变

yarn config get registry

在日益流行前端工程化中，很多同学喜欢使用sass，然而node-sass的下载在国内是个老大难的问题，如果你不翻墙，默认下载极大可能会失败。怎么办呢？ 配置下 node-sass 的二进制包镜像地址就行了

yarn config set sass\_binary\_site http://cdn.npm.taobao.org/dist/node-sass -g

## Vue创建新项目：

1、 cmd窗口进入指定目录下

2、 执行vue init webpack projectName，projectName就是你的项目名称，一路回车搞定

## element-ui Input 标签后加搜索图标

<el-form ref="form" :model="form">

<el-input v-model="form.name" placeholder="请输入名称"

suffix-icon="el-icon-search"></el-input>

</el-form>

suffix-icon 在搜索框后加入图标

prefix-icon 在搜索框头部加入图标

## 记录：js删除数组中某一项或几项的几种方法

1:js中的splice方法

splice(index,len,[item]) 注释：该方法会改变原始数组。

splice有3个参数，它也可以用来替换/删除/添加数组内某一个或者几个值

index:数组开始下标 len: 替换/删除的长度 item:替换的值，删除操作的话 item为空

如：arr = ['a','b','c','d']

删除 ---- item不设置

arr.splice(1,1) //['a','c','d'] 删除起始下标为1，长度为1的一个值，len设置的1，如果为0，则数组不变

arr.splice(1,2) //['a','d'] 删除起始下标为1，长度为2的一个值，len设置的2

替换 ---- item为替换的值

arr.splice(1,1,'ttt') //['a','ttt','c','d'] 替换起始下标为1，长度为1的一个值为‘ttt’，len设置的1

arr.splice(1,2,'ttt') //['a','ttt','d'] 替换起始下标为1，长度为2的两个值为‘ttt’，len设置的1

添加 ---- len设置为0，item为添加的值

arr.splice(1,0,'ttt') //['a','ttt','b','c','d'] 表示在下标为1处添加一项‘ttt’

看来还是splice最方便啦

2：delete delete删除掉数组中的元素后，会把该下标出的值置为undefined,数组的长度不会变

如：delete arr[1] //['a', ,'c','d'] 中间出现两个逗号，数组长度不变，有一项为undefined

还有其他几种自定义方法，参考这里

## 获取下拉选择的值的Label

let obj = {};

obj = this.selectOptions.find((item) => {

return item.value == this.selectedValue;

});

Label的内容为obj.lable

## filter过滤数组

filter()

1.方法创建一个新的数组，新数组中的元素是通过检查指定数组中符合条件的所有元素

2.不会改变原始数组

var arr = [

{ id: 1, text: 'aa', done: true },

{ id: 2, text: 'bb', done: false }

]

console.log(arr.filter(item => item.done))

转成ES5

arr.filter(function (item) {

return item.done;

});

return后面判断结果，取布尔值，true的话就添入新的filter数组中，false的话，不会添进filter的数组中。

最后得到新的数组是

[{id: 1, text: "aa", done: true}]

var arr = [

{ id: 1, text: 'aa', done: true },

{ id: 2, text: 'bb', done: '' }

]

console.log(arr.filter(item => item.done))

结果还是一样，因为”空字符串转成布尔类型为false

用处

1.去掉数组中的空字符串项

var arr = ['1.jpg','2.jpg','', '3.jpg',' ']

var newArr = arr.filter(item => item)

console.log(newArr,arr)

newArr => ["1.jpg", "2.jpg", "3.jpg", " "] 空字符串项被过滤掉了

arr => ["1.jpg", "2.jpg", "", "3.jpg", " "]

2.去掉数组中不符合的项

var arr = [73,84,56, 22,100]

var newArr = arr.filter(item => item>80) //得到新数组 [84, 100]

console.log(newArr,arr)

var arr = ['aa','cb','cc', 'bd','rf']

var newArr = arr.filter(item => item.indexOf('b')<0) 不包含b字符串的

得到新数组 ["aa", "cc", "rf"]

## js 字符串转化成数字

方法主要有三种

转换函数、强制类型转换、利用js变量弱类型转换。

1. 转换函数：

js提供了parseInt()和parseFloat()两个转换函数。前者把值转换成整数，后者把值转换成浮点数。只有对String类型调用这些方法，这两个函数才能正确运行；对其他类型返回的都是NaN(Not a Number)。

一些示例如下：

parseInt("1234blue"); //returns 1234

parseInt("0xA"); //returns 10

parseInt("22.5"); //returns 22

parseInt("blue"); //returns NaN

parseInt()方法还有基模式，可以把二进制、八进制、十六进制或其他任何进制的字符串转换成整数。基是由parseInt()方法的第二个参数指定的，示例如下：

parseInt("AF", 16); //returns 175

parseInt("10", 2); //returns 2

parseInt("10", 8); //returns 8

parseInt("10", 10); //returns 10

如果十进制数包含前导0，那么最好采用基数10，这样才不会意外地得到八进制的值。例如：

parseInt("010"); //returns 8

parseInt("010", 8); //returns 8

parseInt("010", 10); //returns 10

parseFloat()方法与parseInt()方法的处理方式相似。

使用parseFloat()方法的另一不同之处在于，字符串必须以十进制形式表示浮点数，parseFloat()没有基模式。

下面是使用parseFloat()方法的示例：

parseFloat("1234blue"); //returns 1234.0

parseFloat("0xA"); //returns NaN

parseFloat("22.5"); //returns 22.5

parseFloat("22.34.5"); //returns 22.34

parseFloat("0908"); //returns 908

parseFloat("blue"); //returns NaN

2. 强制类型转换

还可使用强制类型转换（type casting）处理转换值的类型。使用强制类型转换可以访问特定的值，即使它是另一种类型的。

ECMAScript中可用的3种强制类型转换如下：

Boolean(value)——把给定的值转换成Boolean型；

Number(value)——把给定的值转换成数字（可以是整数或浮点数）；

String(value)——把给定的值转换成字符串。

用这三个函数之一转换值，将创建一个新值，存放由原始值直接转换成的值。这会造成意想不到的后果。

当要转换的值是至少有一个字符的字符串、非0数字或对象（下一节将讨论这一点）时，Boolean()函数将返回true。如果该值是空字符串、数字0、undefined或null，它将返回false。

可以用下面的代码段测试Boolean型的强制类型转换。

Boolean(""); //false – empty string

Boolean("hi"); //true – non-empty string

Boolean(100); //true – non-zero number

Boolean(null); //false - null

Boolean(0); //false - zero

Boolean(new Object()); //true – object

Number()的强制类型转换与parseInt()和parseFloat()方法的处理方式相似，只是它转换的是整个值，而不是部分值。示例如下：

用　　法 结　　果

Number(false) 0

Number(true) 1

Number(undefined) NaN

Number(null) 0

Number( "5.5 ") 5.5

Number( "56 ") 56

Number( "5.6.7 ") NaN

Number(new Object()) NaN

Number(100) 100

最后一种强制类型转换方法String()是最简单的，示例如下：

var s1 = String(null); //"null"

var oNull = null;

var s2 = oNull.toString(); //won’t work, causes an error

3. 利用js变量弱类型转换

举个小例子，一看，就会明白了。

<script>

var str= '012.345 ';

var x = str-0;

x = x\*1;

</script>

上例利用了js的弱类型的特点，只进行了算术运算，实现了字符串到数字的类型转换，不过这个方法还是不推荐的

## js 获取json对象的Key、value

<script type="text/javascript">

getJson('age');

function getJson(key){

var jsonObj={"name":"傅红雪","age":"24","profession":"刺客"};

//1、使用eval方法

var eValue=eval('jsonObj.'+key);

alert(eValue);

//2、遍历Json串获取其属性

for(var item in jsonObj){

if(item==key){ //item 表示Json串中的属性，如'name'

var jValue=jsonObj[item];//key所对应的value

alert(jValue);

}

}

//3、直接获取

alert(jsonObj[''+key+'']);

}

</script>

## AVUE常见Tips

**组件操作：**

Tab组件：

A.父页面中：

1．引入外部页面作为组件

**import** runRating **from** '@/views/platform/debt/runRating'

2．定义组件

components: {runRating},

data(){

return{

model\_rating: '',  
last\_rating: '',

}

}

3．将组件放入父页面

<el-tab-pane label="外部组件" name="six">  
 <runRating ref="runRating" :last\_rating="last\_rating" :model\_rating="model\_rating"  
 @debtRatingInfo='debtRatingInfo'></runRating>  
</el-tab-pane>

4．调用组件方法

**this**.$refs.runRating.getList(**this**.$route.query.key)

5．父页面接收子页面传值

debtRatingInfo: **function** (data) {  
 console.log(data)   
 **this**.debtRatingList = data[0]  
 **this**.bondCodes = data[1]   
},

B.子页面

1．子页面接收父页面传值（props和method同一级别）

props: ['last\_rating', 'model\_rating'],

2．子页面传值给父页面

**this**.$emit('debtRatingInfo', **this**.debtRatingInfo);

弹窗组件：

1. 父组件

1.引入外部弹窗组件

**import** addWaitRating **from** './addWaitRating'

**import** selectBonds **from** "@/views/platform/debt/selectBonds";

2.定义页面组件

components: {addWaitRating, selectBonds},

3.将组件放入页面

<div>  
 <addWaitRating v-if="visible" @addVisible='addVisible' :companyCode="companyCode"></addWaitRating>

<selectBonds v-if="bondVisible" @selectVisible='selectVisible'  
 :bondCodes="bondCodes"></selectBonds>  
</div>

4.定义弹窗逻辑值默认false不显示

visible: **false**,  
bondVisible: **false**,

5.定义弹窗事件弹窗显示为true，同时可定义传递参数值

handleAdd: **function** () {  
 **this**.key = **this**.$route.query.key;  
 **this**.companyCode = **this**.key  
 **this**.visible = **true**},

selectDebt() {  
 **this**.bondVisible = **true**},

6.接收子组件传值，并将弹窗显示逻辑复原为初始值false

addVisible() {  
 **this**.visible = **false  
 this**.getList(**this**.key)  
},

selectVisible: **function** (data) {  
 console.log(data)   
 **this**.bondCodes = data  
 **this**.bondVisible = **false**  
},

B．子组件：

1.定义弹窗组件

<el-dialog title="添加待评债项" :visible.sync="dialogTableVisible" :before-close="beforeClose" width="80%">…</el-dialog>

<el-dialog title="选择债券列表" :visible.sync="selectVisible" :before-close="beforeClose" width="80%">…</el-dialog>

2.定义弹窗逻辑值默认true显示

ialogTableVisible: **true**,

selectVisible: **true**,

3.接收父组件传值

props: ['companyCode'],

props: ['bondCodes'],

4.关闭弹窗（可不关闭this.selectVisible = false去掉），初始化值并传递给父组件

selectedAdd() {  
 **let** len = **this**.tableDataSelected.length  
 **let** debtCodes = ''  
 **for** (**let** i = 0; i < len; i++) {  
 i == len - 1 ? debtCodes += **this**.tableDataSelected[i].debtCode : debtCodes += ( + **this**.tableDataSelected[i].debtCode + ',')  
 }  
 **this**.selectVisible = **false  
 this**.debtCodeData = debtCodes  
 **this**.$emit('selectVisible', **this**.debtCodeData)  
 // this.$message.success('选中的数据' + JSON.stringify(debtCodes));  
},

5.关闭弹窗前将弹窗显示逻辑复原为初始值true（若没有设置this.selectVisible = false，则不需要），传值给父组件（同上）

beforeClose() {  
 **this**.$emit('addVisible')//不需传值  
},

beforeClose() {  
 **this**.selectVisible = **true  
 this**.$emit('selectVisible', **this**.debtCodeData)  
},

## JavaScript 判断字符串是纯数字

本文链接：https://blog.csdn.net/ligen52/article/details/82187298

方法一:是十进制纯数字

使用正则表达式

const result = /^\d+$/.test(number)

1

方法二:是数字(不仅仅是十进制)

使用 isNaN(num)

isNaN(num) //返回true代表非数字,返回false,代表是有效数字

isNaN(123) // false

isNaN('123') // false

isNaN('1e10000') // false

isNaN('foo') // true

isNaN('10px') // true

## js 怎么把json数据中的日期排序

var data = {

"rows": [{

"name": "张三",

"time": "2011/4/1 0:00:00",

},{

"name": "李四",

"time": "2015/5/6 12:30:00",

},{

"name": "王五",

"time": "2012/10/1 22:10:00",

},{

"name": "赵六",

"time": "2011/9/1 22:10:00",

}]

};

var rows = data.rows;

rows.sort(function(a,b){

return Date.parse(a.time) - Date.parse(b.time);//时间正序

});

for(var i =0,l=rows.length;i<l;i++){

console.log(rows[i].name + " | " + rows[i].time);

}

输出结果：

张三 | 2011/4/1 0:00:00

赵六 | 2011/9/1 22:10:00

王五 | 2012/10/1 22:10:00

李四 | 2015/5/6 12:30:00

同理：日期数组排序

var tradingDays = ["2019-07-10", "2019-07-12", "2019-07-11", "2019-07-17", "2019-07-14", "2019-07-15", "2019-07-16", "2019-07-13"]

tradingDays.sort(function (a, b) {

return Date.parse(a) - Date.parse(b);//时间正序

});

## Javascript获取数组中最大和最小值

取出数组中最大值或最小值是开发中常见的需求，今天继续讲解如何获取javascript数组中最大和最小值。

1.排序法

首先我们给数组进行排序，可以按照从小到大的顺序来排，排序之后的数组中第一个和最后一个就是我们想要获取的最小值和最大值。

排序我们会用到数组的 sort 方法。

var arr = [12,56,25,5,82,51,22];

arr.sort(function (a, b) {

return a-b;

}); // [5,12,22,25,51,56]

var min = arr[0]; // 5

var max = arr[arr.length - 1]; // 56

2.假设法

假设当前数组中的第一个值是最大值，然后拿这个最大值和后面的项逐一比较，如果后面的某一个值比假设的值还大，说明假设错了，我们把假设的值进行替换。最后得到的结果就是我们想要的。

// 获取最大值：

var arr = [22,13,6,55,30];

var max = arr[0];

for(var i = 1; i < arr.length; i++) {

var cur = arr[i];

cur > max ? max = cur : null

}

console.log(max); // 55

// 获取最小值：

var arr = [22,13,6,55,30];

var min = arr[0];

for(var i = 1; i < arr.length; i++) {

var cur = arr[i];

cur < min ? min = cur : null

}

console.log(min) // 6

3. 使用 Math 中的 max/min 方法

可以使用apply来实现。apply传入的是一个数组。

var arr = [22,13,6,55,30];

var max = Math.max.apply(null, arr);

var min = Math.min.apply(null, arr);

console.log(max, min) // 55,6

4. 使用ES6的扩展运算符

var arr = [22,13,6,55,30];

console.log(arr.Math(...arr)); // 55

## avue屏幕设置

export default {

getScreen: function () {

//获取客户端屏幕宽度

var width = document.body.clientWidth

if (width >= 1200) {

return 3 // 大屏幕

} else if (width >= 992) {

return 2 // 中屏幕

} else if (width >= 768) {

return 1 // 小屏幕

} else {

return 0 // 超小屏幕

}

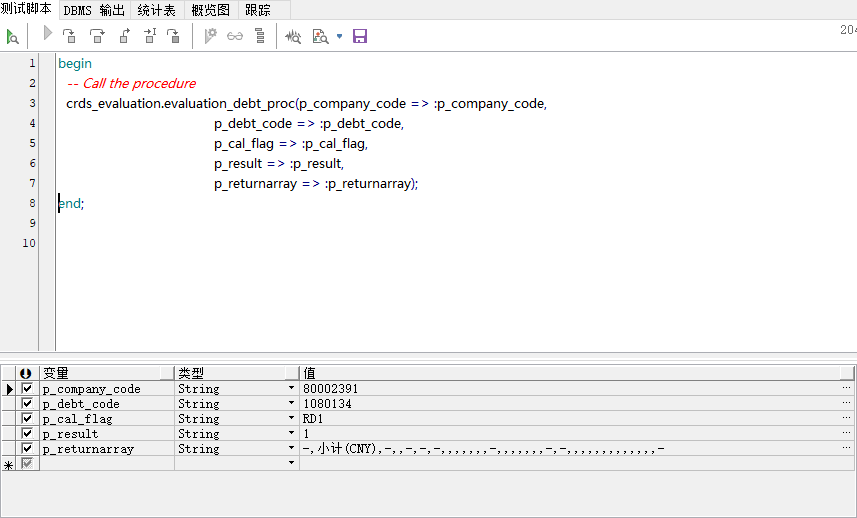
}

}

# 数据库知识技巧

## PL/SQL 调用oracle存储过程方法

1. 右键存储过程-->测试(test):



在下面的变量列表里填写对应变量值，运行

1. SQL语句调用

DECLARE

errcode VARCHAR2(1000);

returnString CLOB;

BEGIN CRDS\_EVALUATION.EVALUATION\_DEBT\_PROC('80084396','136398','RD1',errcode,returnString);

dbms\_output.put\_line('查询到的结果为： '||returnString);

END;

## Mysql中常用的几种时间类型

date、datetime、time、year、timestamp

Datetime : 时间日期型，格式是YYYY-mm-dd HH:ii:ss，表示的范围是从1000到9999。但是有零值，0000-00-00 00:00:00；

Date：日期，就是datetime中的date部分；

Time：时间(段)，指定的某个区间之间，从-时间到+时间(有负时间表示)；

Timestamp：时间戳，并不是常规意义时间戳(如：14253685)，范围是’1970-01-01 00:00:00’到2037年。YYYY-mm-dd HH:ii:ss,格式与datetime完全一致；

Year：yy和yyyy。yyyy的范围是1901~2155，yy的范围是1970~2069

两位year(00~69表示2000~2069,70~99表示1970~1999)。当应用只需要记录年份时，year比date更省空间

date : 只有日期，没有时间；

datetime ： 有时间，有日期；

time ： 只有时间 ，精确到分秒 ；

timestamp ： 时间戳，精确到分秒；

year ： 年，如2002，如果写为 2002-01-15，将会进行计算，插入结果为1986

DATE\_FORMAT( )

可以使用date\_format( )函数进行时间的转换。

格式 描述

%a 缩写星期名

%b 缩写月名

%c 月，数值

%D 带有英文前缀的月中的天

%d 月的天，数值(00-31)

%e 月的天，数值(0-31)

%f 微秒

%H 小时 (00-23)

%h 小时 (01-12)

%I 小时 (01-12)

%i 分钟，数值(00-59)

%j 年的天 (001-366)

%k 小时 (0-23)

%l 小时 (1-12)

%M 月名

%m 月，数值(00-12)

%p AM 或 PM

%r 时间，12-小时（hh:mm:ss AM 或 PM）

%S 秒(00-59)

%s 秒(00-59)

%T 时间, 24-小时 (hh:mm:ss)

%U 周 (00-53) 星期日是一周的第一天

%u 周 (00-53) 星期一是一周的第一天

%V 周 (01-53) 星期日是一周的第一天，与 %X 使用

%v 周 (01-53) 星期一是一周的第一天，与 %x 使用

%W 星期名

%w 周的天 （0=星期日, 6=星期六）

%X 年，其中的星期日是周的第一天，4 位，与 %V 使用

%x 年，其中的星期一是周的第一天，4 位，与 %v 使用

%Y 年，4 位

%y 年，2 位

str\_to\_date

字符串转换为date

str\_to\_date(

'2016-12-15 16:48:40',

'%Y-%m-%d %H:%i:%S'

)

select str\_to\_date('31-09-2016','%d-%m-%Y'); 结果：2016-09-31

获取mysql系统时间的三种方法

1、select now() --yyyy-MM-dd HH:mm:ss

2、select sysdate() --yyyy-MM-dd HH:mm:ss

3、select current\_date --yyyy-MM-dd

## 数据库连接数设置

ORA-12519, TNS:no appropriate service handler found The Connection descriptor used by the client waslocalhost:1521:orcl

发现问题原因：

在用数据库连接工具连接数据库的时候，提示：ORA-12519 TNS:no appropriate service handler found。根据网上给的解决方法，用sqlplus / as sysdba连接数据库，提示：ORA-00020: maximum number of processes (150) exceeded

解决方案：

1、先关闭应用与数据库的连接，然后用sqlplus / as sysdba 连接数据库

2、SQL>show parameter processes

aq\_tm\_processes integer 1

db\_writer\_processes integer 3

gcs\_server\_processes integer 0

global\_txn\_processes integer 1

job\_queue\_processes integer 1000

log\_archive\_max\_processes integer 4

processes integer 150

SQL> alter system set processes=500 scope=spfile;

SQL>shutdown immediate;

SQL>startup;

修改系统参数，重启数据库。

SQL> show parameter processes

aq\_tm\_processes integer 1

db\_writer\_processes integer 3

gcs\_server\_processes integer 0

global\_txn\_processes integer 1

job\_queue\_processes integer 1000

log\_archive\_max\_processes integer 4

processes integer 500

至此，增大了数据库最大连接数，服务器应用重新连接，本地工具也可以连接使用了。

## SQL> shutdown immediate;

shutdown immediate

ORA-00900: 无效 SQL 语句

shutdown immediate命令

PL/SQL Developer Commmand and SQL Windows 是不可以的。

需要用cmd sqlplus

Microsoft Windows XP [版本 5.1.2600]

(C) 版权所有 1985-2001 Microsoft Corp.

D:\>sqlplus /nolog

SQL\*Plus: Release 9.2.0.1.0 - Production on 星期五 2月 9 10:57:22 2007

Copyright (c) 1982, 2002, Oracle Corporation. All rights reserved.

SQL> connect sys/sys as sysdba;

已连接。

SQL> shutdown immediate;

数据库已经关闭。

已经卸载数据库。

ORACLE 例程已经关闭。

SQL>

## Oracle表字段的增加、删除、修改和重命名

增加字段语法：alter table tablename add (column datatype [default value][null/not null],….);

说明：alter table 表名 add (字段名 字段类型 默认值 是否为空);

例：alter table sf\_users add (HeadPIC blob);

例：alter table sf\_users add (userName varchar2(30) default '空' not null);

修改字段的语法：alter table tablename modify (column datatype [default value][null/not null],….);

说明：alter table 表名 modify (字段名 字段类型 默认值 是否为空);

例：alter table sf\_InvoiceApply modify (BILLCODE number(4));

删除字段的语法：alter table tablename drop (column);

说明：alter table 表名 drop column 字段名;

例：alter table sf\_users drop column HeadPIC;

字段的重命名：

说明：alter table 表名 rename column 列名 to 新列名 （其中：column是关键字）

例：alter table sf\_InvoiceApply rename column PIC to NEWPIC;

表的重命名：

说明：alter table 表名 rename to 新表名

例：alter table sf\_InvoiceApply rename to sf\_New\_InvoiceApply;

## mybatis中使用IN函数

//listproductSn 是一个list集合或者array数组，其存储函数IN所需的条件

<select id=”selectIn”>

select \* from A where 1=1 AND product\_sprotType IN

<foreach collection="listproductSn" item="item" index="index" open="(" separator="," close=")">

#{item}

</foreach>

</select>

## 查询数据库用户所有字段

select \* from user\_tab\_columns a where a.column\_name like '%company%code%'

## Mybatis-Plus基本使用

本文链接：https://blog.csdn.net/qq\_33799320/article/details/83538815

官网:http://mp.baomidou.com/guide/

条件

wrapper.between(column, val1, val2)

wrapper.groupBy(columns) //对应sql中分组

wrapper.eq(column, params) //相当于where条件

wrapper.in(column, value) //sql中in

wrapper.notIn(column, value) //sql中 not in

wrapper.orderBy(columns, isAsc) //排序

wrapper.exists(value) //相对于sql中exists查询

wrapper.notExists(value) //相当于sql中not exists查询

wrapper.notBetween(column, val1, val2) //相当于sql中在某个范围内使用的between

wrapper.ge(column, params) //大于等于

wrapper.le(column, params) //小于等于

wrapper.like(column, value) //模糊查询

wrapper.having(sqlHaving, params) //条件

查询

List<UserEntity> list = ud.selectByMap(columnMap) //通过map查询 适用场景是分页查询

List<UserEntity> list2 = ud.selectList(wrapper); //如果为Null，默认查询所有

List<UserEntity> list3 = ud.selectCount(wrapper)//通过EntityWrapper根据某个字段获取总数

List<UserEntity> list4 = ud.selectMaps(wrapper) //EntityWrapper查询

List<UserEntity> list5 = ud.selectPage(rowBounds, wrapper) //rowBounds里面封装了起止和长度，wrapper具体

增删改

ud.insert(entity) //应用场景是指定你需要的实体增加到对应的表中

ud.delete(wrapper) //通过EntityWrapper指定字段删除数据

ud.deleteById(id) //主键删除

ud.deleteByMap(columnMap) //通过map删除

ud.deleteBatchIds(idList) //批量删除

ud.insertAllColumn(entity) //默认插入该实体所有字段

ud.insertUserEntity(map) //通过map新增数据

ud.update(entity, wrapper) //通过EntityWrapper更新实体

ud.updateAllColumnById(entity) //更新所有实体,通过主键更新

ud.updateById(entity) //更新指定实体,通过主键更新

ud.updateUserMap(map) //通过map

条件构造器

AbstractWrapper条件参数

QueryWrapper(LambdaQueryWrapper) 和 UpdateWrapper(LambdaUpdateWrapper) 的父类

用于生成 sql 的 where 条件, entity 属性也用于生成 sql 的 where 条件

查询方式 说明

where WHERE 语句，拼接 + WHERE 条件

and AND 语句，拼接 + AND 字段=值

or OR 语句，拼接 + OR 字段=值 !!!主动调用or表示紧接着下一个方法不是用and连接!(不调用or则默认为使用and连接)

eq 等于=

allEq 基于 map 内容等于=

ne 不等于<>

gt 大于>

ge 大于等于>=

lt 小于<

le 小于等于<=

like 模糊查询 LIKE

likeLeft LIKE ‘%值’

likeRight LIKE ‘值%’

notLike 模糊查询 NOT LIKE

in IN (v0, v1, …)

notIn NOT IN (v0, v1, …)

inSql IN ( sql语句 )

notInSql 字段 NOT IN ( sql语句 )

isNull NULL 值查询

isNotNull IS NOT NULL

groupBy GROUP BY 字段, …

having HAVING ( sql语句 )

orderBy ORDER BY 字段, …

orderByAsc ORDER BY 字段, … ASC

orderByDesc ORDER BY 字段, … DESC

exists EXISTS 条件语句

notExists NOT EXISTS 条件语句

between BETWEEN 条件语句

notBetween NOT BETWEEN 条件语句

addFilter 自由拼接 SQL

last 无视优化规则直接拼接到 sql 的最后 !!!只能调用一次,多次调用以最后一次为准 有sql注入的风险,请谨慎使用

apply 拼接SQL

QueryWrapper条件

说明:继承自 AbstractWrapper ,自身的内部属性 entity 也用于生成 where 条件

及 LambdaQueryWrapper, 可以通过 new QueryWrapper().lambda() 方法获取

查询方式 说明

select 设置查询字段

UpdateWrapper条件

说明:继承自 AbstractWrapper ,自身的内部属性 entity 也用于生成 where 条件

及 LambdaUpdateWrapper, 可以通过 new UpdateWrapper().lambda() 方法获取!

查询方式 说明

set SQL SET 字段

setSql 设置 SET 部分 SQL

分页

if (search != null && search.size() > 0) {

for (String key : search.keySet()) {

String value = search.get(key);

queryWrapper.like(key, value);

}

}

IPage<PartsSpu> iPage = partsSpuMapper.selectPage(new Page<PartsSpu>(page, limit), queryWrapper);

List<PartsSpu> records = iPage.getRecords();

return this.getIPageApiResult(iPage, records);

简单的模糊查询

// 对应SQL: SELECT \* FROM SYS\_USER WHERE USERNAME LIKE '%O%'

public List<SysUser> QueryWrapper(String column, String value) {

Wrapper<SysUser> userWrapper = new QueryWrapper<SysUser>();

QueryWrapper<SysUser> like = ((QueryWrapper<SysUser>) userWrapper).like("username", "o");

return userMapper.selectList(like);

};

多条件拼接

//对应sql:select username from sys\_user where id > 1

public List<SysUser> query(){

return userMapper.selectList(

new QueryWrapper<SysUser>()

.select("username")

.gt("id",1)

);

}

## 创建视图、查看视图、修改视图及删除视图的语法命令讲解

视图是一张虚拟的表，对视图中数据的增、删、改、查和数据表一样。简单来说，视图就是对多张表进行SELECT查询的结果集。

关于视图和数据表的关系，有以下几个要点：

视图是一种虚拟的表，本身并不独自占用物理空间，它是由创建它的数据表中的数据而形成的一个抽象的数据集合，是数据表中数据的子集。

视图和创建它的数据表共享相同字段的值，也就是说，数据表中字段的值若发生变化，视图中对应的值也会发生改变，反之亦然。

视图的操作和数据表的操作方式类似，也可以对视图中的记录进行增、删、改、查。用的最多的是查询操作。

对视图中记录的增、删、改，会影响对应的数据表中的记录。但删除整个视图，不会对数据表产生影响。

多表关联查询的时候，可以将多表关联查询的结果作为一个视图，以后只需从视图中进行查询即可，而不用每次都进行多表查询。

1. 创建视图

基本语法：

CREATE [OR REPLACE] VIEW view\_name AS select\_statement

如果写了OR REPLACE，当视图view\_name已经存在时，就会替换已有的视图。

create view v\_user as select id,username from user;

-- 创建视图v\_user

2. 查看视图

查看所有的视图：

SHOW TABLES;

-- 查看所有的数据表和视图

SHOW TABLES LIKE 'v\_%';

-- 模糊查询

查看视图的详细信息：

SHOW CREATE VIEW v\_user

3. 修改视图

基本语法：

ALTER [OR REPLACE] VIEW 视图名称 AS select\_statement

4. 删除视图

DROP VIEW v\_user

## Oracle sysdate截取年月日

使用trunc函数：

select trunc(sysdate) from dual;

从timestamp中获取年月日时分秒

select

extract(year from systimestamp) year

,extract(month from systimestamp) month

,extract(day from systimestamp) day

,extract(minute from systimestamp) minute

,extract(second from systimestamp) second

,extract(timezone\_hour from systimestamp) th

,extract(timezone\_minute from systimestamp) tm

,extract(timezone\_region from systimestamp) tr

,extract(timezone\_abbr from systimestamp) ta

from dual

## 分组排序-分组统计

select \* from

(select t.\*, row\_number() over (partition by column\_name1 order by column\_name2 desc) rn ,count(\*) over (partition by column\_name3) nums from table\_name t)

 where rn=1

通过column\_name1分组column\_name2排序，以column\_name3分组统计

## 分组结果合并字符串

select column\_name1,listagg(column\_name2,'，') within GROUP (order by column\_name3) alias

from table\_name

where del\_flag = '0' group by column\_name1

通过column\_name1分组查询后将符合条件的结果中的column\_name2字段以column\_name3排序用逗号分隔拼接成字符串

## Oracle查询结果行转列

select \*from (select t. column\_name1, t. column\_name2,t. column\_name3, t. column\_name4

from table\_name t)

pivot (max(column\_name4)

for column\_name3 in (' column\_name3\_value1' alias1 , ' column\_name3\_value2' alias2 ,

' column\_name3\_value3' alias3 , …… ) )

从table中获取记录，将记录指定字段column\_name3的值' column\_name3\_value1' alias1 , ' column\_name3\_value2' alias2 , ' column\_name3\_value3' alias3 , ……作为列字段名，指定对应字段column\_name4的值作为新列字段对应值，形成行转列

## Oracle merge into用法

本文链接：https://blog.csdn.net/qq\_40992849/article/details/80985417

从备份表中更新字段到正式表中，使用 UPDATE 批量更新大量的数据，会出现效率低下，有时候甚至卡死的情况，后面通过使用 MERGE INTO 代替 UPDATE 执行批量更新，会提升执行效率。

MERGE INTO语法如下：

MERGE INTO table\_name alias1

USING (table|view|sub\_query) alias2

ON (join condition)

WHEN MATCHED THEN

UPDATE

SET col1 = col1\_val1,

col2 = col2\_val2

WHEN NOT MATCHED THEN

INSERT (column\_list) VALUES (column\_values);

其中，table\_name 指的是更新的表，using()里边的指的是数据来源表/视图/子查询结果集，condition指的是连接条件，如果满足连接条件，set 字段1=值1，字段2=值2...

如果连接条件不满足，则停止更新进行插入。

MERGE INTO merge\_target target

USING (SELECT B.name,B.age,B.target\_id FROM merge\_source B) source

ON (target.id=source.target\_id)

WHEN MATCHED THEN

UPDATE

SET target.name = source.name,

target.age = source.age

WHEN NOT MATCHED THEN

INSERT(target.name,target.age) VALUES (source.name,source.age);

# JAVA小知识点

## 类型转换

java中的字符，字符串，数字之间的转换

string 和int之间的转换

string转换成int :Integer.valueOf("12")

int转换成string : String.valueOf(12)

char和int之间的转换

首先将char转换成string

String str=String.valueOf('2')

Integer.valueof(str) 或者Integer.PaseInt(str)

Integer.valueof返回的是Integer对象，Integer.paseInt返回的是int

## Mybatis 中#{} 和${} 的区别：

1. #将传入的数据都当成一个字符串，会对自动传入的数据加一个双引号。如：order by #user\_id#，如果传入的值是111,那么解析成sql时的值为order by "111", 如果传入的值是id，则解析成的sql为order by "id".

2. $将传入的数据直接显示生成在sql中。如：order by $user\_id$，如果传入的值是111,那么解析成sql时的值为order by user\_id, 如果传入的值是id，则解析成的sql为order by id.

3. #方式能够很大程度防止sql注入。

4.$方式无法防止Sql注入

5.$方式一般用于传入数据库对象，例如传入表名.

6.一般能用#的就别用$.

## Java 8 中 Date与LocalDateTime、LocalDate、LocalTime互转

Java 8中 java.util.Date 类新增了两个方法，分别是from(Instant instant)和toInstant()方法

// Obtains an instance of Date from an Instant object.

public static Date from(Instant instant) {

try {

return new Date(instant.toEpochMilli());

} catch (ArithmeticException ex) {

throw new IllegalArgumentException(ex);

}

}

// Converts this Date object to an Instant.

public Instant toInstant() {

return Instant.ofEpochMilli(getTime());

}

这两个方法使我们可以方便的实现将旧的日期类转换为新的日期类，具体思路都是通过Instant当中介，然后通过Instant来创建LocalDateTime（这个类可以很容易获取LocalDate和LocalTime），新的日期类转旧的也是如此，将新的先转成LocalDateTime，然后获取Instant，接着转成Date，具体实现细节如下：

// 01. java.util.Date --> java.time.LocalDateTime

public void UDateToLocalDateTime() {

java.util.Date date = new java.util.Date();

Instant instant = date.toInstant();

ZoneId zone = ZoneId.systemDefault();

LocalDateTime localDateTime = LocalDateTime.ofInstant(instant, zone);

}

// 02. java.util.Date --> java.time.LocalDate

public void UDateToLocalDate() {

java.util.Date date = new java.util.Date();

Instant instant = date.toInstant();

ZoneId zone = ZoneId.systemDefault();

LocalDateTime localDateTime = LocalDateTime.ofInstant(instant, zone);

LocalDate localDate = localDateTime.toLocalDate();

}

// 03. java.util.Date --> java.time.LocalTime

public void UDateToLocalTime() {

java.util.Date date = new java.util.Date();

Instant instant = date.toInstant();

ZoneId zone = ZoneId.systemDefault();

LocalDateTime localDateTime = LocalDateTime.ofInstant(instant, zone);

LocalTime localTime = localDateTime.toLocalTime();

}

// 04. java.time.LocalDateTime --> java.util.Date

public void LocalDateTimeToUdate() {

LocalDateTime localDateTime = LocalDateTime.now();

ZoneId zone = ZoneId.systemDefault();

Instant instant = localDateTime.atZone(zone).toInstant();

java.util.Date date = Date.from(instant);

}

// 05. java.time.LocalDate --> java.util.Date

public void LocalDateToUdate() {

LocalDate localDate = LocalDate.now();

ZoneId zone = ZoneId.systemDefault();

Instant instant = localDate.atStartOfDay().atZone(zone).toInstant();

java.util.Date date = Date.from(instant);

}

// 06. java.time.LocalTime --> java.util.Date

public void LocalTimeToUdate() {

LocalTime localTime = LocalTime.now();

LocalDate localDate = LocalDate.now();

LocalDateTime localDateTime = LocalDateTime.of(localDate, localTime);

ZoneId zone = ZoneId.systemDefault();

Instant instant = localDateTime.atZone(zone).toInstant();

java.util.Date date = Date.from(instant);

}

## 日期格式转换

形如Mon Dec 31 00:00:00 CST 2012字符串转换为相应格式的日期字符串

public static void formatDate(Date date) {

SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd");

String dateFormat = simpleDateFormat.format(date);

Date formatDate = simpleDateFormat.parse(dateFormat);

System.out.println(dateFormat);

System.out.println(formatDate);

}

## 实体类表字段注解

@JsonSerialize(using = ToStringSerializer.class)

：数字类型字段前后传递防止损失精度

@TableField(exist=false)

:实体类中非表字段

## Java计算时间差、日期差总结

1、java 7中的日历类Calendar

Calendar类使用其静态的getInstance()方法获取一个日历实例，该实例为当前的时间；如果想改变时间，可以通过其setTime方法传入一个Date对象，即可获得Date对象所表示时间的Calendar对象

/\*\*

\*使用Calendar对象计算时间差，可以按照需求定制自己的计算逻辑

\* @param strDate

\* @throws ParseException

\*/

public static void calculateTimeDifferenceByCalendar(String strDate) throws ParseException {

SimpleDateFormat formatter = new SimpleDateFormat("yyyy/MM/dd");

Date date = formatter.parse(strDate);

Calendar c1 = Calendar.getInstance(); //当前日期

Calendar c2 = Calendar.getInstance();

c2.setTime(date); //设置为另一个时间

int year = c1.get(Calendar.YEAR);

int oldYear = c2.get(Calendar.YEAR);

//这里只是简单的对两个年份数字进行相减，而没有考虑月份的情况

System.out.println("传入的日期与今年的年份差为：" + (year - oldYear));

}

输出为：

传入的日期与今年的年份差为：12

1

2、java 8中的周期类Period

通过调用Period类的静态方法between，传入两个待比较的LocalDate对象today与oldDate，得到的Period的对象p中就包含了today与oldDate两个日期相差的年、月、日信息，可以通过p.getYears()等方法取出

/\*\*

\* 使用java 8的Period的对象计算两个LocalDate对象的时间差，严格按照年、月、日计算，如：2018-03-12 与 2014-05-23 相差 3 年 9 个月 17 天

\* @param year

\* @param month

\* @param dayOfMonth

\*/

public static void calculateTimeDifferenceByPeriod(int year, Month month, int dayOfMonth) {

LocalDate today = LocalDate.now();

System.out.println("Today：" + today);

LocalDate oldDate = LocalDate.of(year, month, dayOfMonth);

System.out.println("OldDate：" + oldDate);

Period p = Period.between(oldDate, today);

System.out.printf("目标日期距离今天的时间差：%d 年 %d 个月 %d 天\n", p.getYears(), p.getMonths(), p.getDays());

}

输出为：

Today：2018-03-13

OldDate：2014-05-23

目标日期距离今天的时间差：3 年 9 个月 18 天

3、java 8中的Duration类

Duration与Period相对应，Period用于处理日期，而Duration计算时间差还可以处理具体的时间，也是通过调用其静态的between方法，该方法的签名是between(Temporal startInclusive, Temporal endExclusive)，因此可以传入两个Instant的实例（Instant实现了Temporal接口），并可以以毫秒（toMillis）、秒（getSeconds）等多种形式表示得到的时间差

public static void calculateTimeDifferenceByDuration() {

Instant inst1 = Instant.now(); //当前的时间

System.out.println("Inst1：" + inst1);

Instant inst2 = inst1.plus(Duration.ofSeconds(10)); //当前时间+10秒后的时间

System.out.println("Inst2：" + inst2);

Instant inst3 = inst1.plus(Duration.ofDays(125)); //当前时间+125天后的时间

System.out.println("inst3：" + inst3);

System.out.println("以毫秒计的时间差：" + Duration.between(inst1, inst2).toMillis());

System.out.println("以秒计的时间差：" + Duration.between(inst1, inst3).getSeconds());

}

输出为：

Inst1：2018-03-13T09:06:00.691Z

Inst2：2018-03-13T09:06:10.691Z

inst3：2018-07-16T09:06:00.691Z

以毫秒计的时间差：10000

以秒计的时间差：10800000

1

2

3

4

5

4、java 8中的ChronoUnit类

ChronoUnit的between方法签名为，between(Temporal temporal1Inclusive, Temporal temporal2Exclusive)，需要注意的是，如果要以不同的单位展示时间差，between入参中的时间对象必须包含有对应的时间信息，否则会抛出java.time.temporal.UnsupportedTemporalTypeException: Unsupported unit XXX的异常

/\*\*

\* 使用java 8的ChronoUnit，ChronoUnit可以以多种单位（基本涵盖了所有的，看源码发现竟然还有“FOREVER”这种单位。。）表示两个时间的时间差

\*/

public static void calculateTimeDifferenceByChronoUnit() {

LocalDate startDate = LocalDate.of(2003, Month.MAY, 9);

System.out.println("开始时间：" + startDate);

LocalDate endDate = LocalDate.of(2015, Month.JANUARY, 26);

System.out.println("结束时间：" + endDate);

long daysDiff = ChronoUnit.DAYS.between(startDate, endDate);

System.out.println("两个时间之间的天数差为：" + daysDiff);

// long hoursDiff = ChronoUnit.HOURS.between(startDate, endDate); //这句会抛出异常，因为LocalDate表示的时间中不包含时分秒等信息

}

输出为：

开始时间：2003-05-09

结束时间：2015-01-26

两个时间之间的天数差为：4280

1

2

3

5、传统的SimpleDateFormat类

用SimpleDateFormat计算时间差的方法，网上找了一份，自己跑了一遍，可以使用，贴在下面

/\*\*

\* 用SimpleDateFormat计算时间差

\* @throws ParseException

\*/

public static void calculateTimeDifferenceBySimpleDateFormat() throws ParseException {

SimpleDateFormat simpleFormat = new SimpleDateFormat("yyyy-MM-dd hh:mm");

/\*天数差\*/

Date fromDate1 = simpleFormat.parse("2018-03-01 12:00");

Date toDate1 = simpleFormat.parse("2018-03-12 12:00");

long from1 = fromDate1.getTime();

long to1 = toDate1.getTime();

int days = (int) ((to1 - from1) / (1000 \* 60 \* 60 \* 24));

System.out.println("两个时间之间的天数差为：" + days);

/\*小时差\*/

Date fromDate2 = simpleFormat.parse("2018-03-01 12:00");

Date toDate2 = simpleFormat.parse("2018-03-12 12:00");

long from2 = fromDate2.getTime();

long to2 = toDate2.getTime();

int hours = (int) ((to2 - from2) / (1000 \* 60 \* 60));

System.out.println("两个时间之间的小时差为：" + hours);

/\*分钟差\*/

Date fromDate3 = simpleFormat.parse("2018-03-01 12:00");

Date toDate3 = simpleFormat.parse("2018-03-12 12:00");

long from3 = fromDate3.getTime();

long to3 = toDate3.getTime();

int minutes = (int) ((to3 - from3) / (1000 \* 60));

System.out.println("两个时间之间的分钟差为：" + minutes);

}

输出为：

两个时间之间的天数差为：11

两个时间之间的小时差为：264

两个时间之间的分钟差为：15840

1

2

3

总结

传统的SimpleDateFormat和Java 7中的Calendar在使用的时候需要自己写一个计算时间差的逻辑，比较麻烦，但是却比较灵活，方便根据自己具体的需求来定制（比如，我想两个日期的天数差15天就算满一个月，不满15天不算一个月，如2018-01-04到2018-02-20，算2个月的时间差）；而Java 8中的几个计算时间差的类更加方便、精确，可以以不同的单位表示得到的时间差，但要注意几个类所包含的时间信息的区别：

System.out.println(LocalDate.now()); //只包含日期信息

System.out.println(LocalTime.now()); //只包含时间信息

System.out.println(LocalDateTime.now()); //包含日期、时间信息

1

2

3

输出为：

2018-03-13

17:13:26.134

2018-03-13T17:13:26.135

1

2

3

以上总结的几个方法只是个例子，具体使用的时候可能需要传入一个或者两个时间进行比较，可能会涉及到这些时间对象的相互转换，Instant、Date、LocalDate等等。。。我就不列举了。。。

另外在使用SimpleDateFormat对String类型的日期进行parse的时候，如果传入的日期为：2017-08-60，这种错误的日期，Java默认会按照日期的信息对其进行转换，formatter.parse("2017-08-60");，得到的日期为2017-09-29，而如果不想进行这种转换，而直接将其判定为输入错误，则可以设置formatter.setLenient(false);，这时就会抛出java.text.ParseException异常了

SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");

formatter.setLenient(false);

try {

Date date = formatter.parse("2017-08-60"); //抛出转换异常

System.out.println(date);

} catch (ParseException e) {

e.printStackTrace();

}

原文：https://blog.csdn.net/sy793314598/article/details/79544796

版权声明：本文为博主原创文章，转载请附上博文链接！

## jackson反序列化失败InvalidFormatException: Can not deserialize value of type java.util.Date

时间:2018-12-25

本文章向大家介绍jackson反序列化失败InvalidFormatException: Can not deserialize value of type java.util.Date，主要包括jackson反序列化失败InvalidFormatException: Can not deserialize value of type java.util.Date使用实例、应用技巧、基本知识点总结和需要注意事项，具有一定的参考价值，需要的朋友可以参考一下。

问题：

客户端使用fastjson进行序列化，服务端使用jackson反序列化失败。报错。

Caused by: com.fasterxml.jackson.databind.exc.InvalidFormatException: Can not deserialize value of type java.util.Date from String "2018-12-25 10:07:49": not a valid representation (error: Failed to parse Date value '2018-12-25 10:07:49': Can not parse date "2018-12-25 10:07:49Z": while it seems to fit format 'yyyy-MM-dd'T'HH:mm:ss.SSS'Z'', parsing fails (leniency? null))

原因

fastjson默认使用的序列化格式：

public static String DEFFAULT\_DATE\_FORMAT = "yyyy-MM-dd HH:mm:ss";

jackson默认支持的日期反序列格式：

("yyyy-MM-dd'T'HH:mm:ss.SSSZ", "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'", "EEE, dd MMM yyyy HH:mm:ss zzz", "yyyy-MM-dd"))

jackson默认的日期反序列不支持yyyy-MM-dd HH:mm:ss 这种格式，所以反序列化失败

解决办法一：重写jackson反序列化

import com.fasterxml.jackson.core.JsonParser;

import com.fasterxml.jackson.core.JsonProcessingException;

import com.fasterxml.jackson.databind.DeserializationContext;

import com.fasterxml.jackson.databind.JsonDeserializer;

import java.io.IOException;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

/\*\*

\* @version 1.0.0

\* @description 重写反序列化

\* @date 2018/12/25 9:51

\*\*/

public class CustomJsonDateDeserializer extends JsonDeserializer<Date> {

@Override

public Date deserialize(JsonParser jp, DeserializationContext ctxt) throws IOException, JsonProcessingException {

SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

String date = jp.getText();

try {

return format.parse(date);

} catch (ParseException e) {

throw new RuntimeException(e);

}

}

}

在需要的date属性上添加注解：

@JsonDeserialize(using = CustomJsonDateDeserializer.class)

private Date txnTime;

使用这种方式，需要重写上面反序列化抽象类

解决办法二：直接使用jackson注解，不需要重写反序列化

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

private Date txnTime;

GMT+8表示时区，东八区

解决办法三：直接使用fastjson注解，不要重写反序列化

@JSONField(format = “yyyy-MM-dd HH:mm:ss”)

private Date txnTime;

添加依赖：

<!-- fastjson -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.2.32</version>

</dependency>

<!-- jackson-->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.6.4</version>

</dependency>

## 判断BigDecimal对象amt值是否等于0

amt.equals(BigDecimal.ZERO)

## java 去掉字符串后面的0

public static void main(String[] args) {

String str = "00v0001000022000002000000000000";

String t = str.replaceAll("0+$", "");

// 成对去掉0

String t = str.replaceAll("(00){1,}$", "%");

}

附 ： 常用的正则表达式：

匹配特定数字：

^[1-9]d\*$　 　 //匹配正整数

^-[1-9]d\*$ 　 //匹配负整数

^-?[1-9]d\*$　　 //匹配整数

^[1-9]d\*|0$　 //匹配非负整数（正整数 + 0）

^-[1-9]d\*|0$　　 //匹配非正整数（负整数 + 0）

^[1-9]d\*.d\*|0.d\*[1-9]d\*$　　 //匹配正浮点数

^-([1-9]d\*.d\*|0.d\*[1-9]d\*)$　 //匹配负浮点数

^-?([1-9]d\*.d\*|0.d\*[1-9]d\*|0?.0+|0)$　 //匹配浮点数

^[1-9]d\*.d\*|0.d\*[1-9]d\*|0?.0+|0$　　 //匹配非负浮点数（正浮点数 + 0）

^(-([1-9]d\*.d\*|0.d\*[1-9]d\*))|0?.0+|0$　　//匹配非正浮点数（负浮点数 + 0）

评注：处理大量数据时有用，具体应用时注意修正

匹配特定字符串：

^[A-Za-z]+$　　//匹配由26个英文字母组成的字符串

^[A-Z]+$　　//匹配由26个英文字母的大写组成的字符串

^[a-z]+$　　//匹配由26个英文字母的小写组成的字符串

^[A-Za-z0-9]+$　　//匹配由数字和26个英文字母组成的字符串

^w+$　　//匹配由数字、26个英文字母或者下划线组成的字符串

在使用RegularExpressionValidator验证控件时的验证功能及其验证表达式介绍如下:

只能输入数字：“^[0-9]\*$”

只能输入n位的数字：“^d{n}$”

只能输入至少n位数字：“^d{n,}$”

只能输入m-n位的数字：“^d{m,n}$”

只能输入零和非零开头的数字：“^(0|[1-9][0-9]\*)$”

只能输入有两位小数的正实数：“^[0-9]+(.[0-9]{2})?$”

只能输入有1-3位小数的正实数：“^[0-9]+(.[0-9]{1,3})?$”

只能输入非零的正整数：“^+?[1-9][0-9]\*$”

只能输入非零的负整数：“^-[1-9][0-9]\*$”

只能输入长度为3的字符：“^.{3}$”

只能输入由26个英文字母组成的字符串：“^[A-Za-z]+$”

只能输入由26个大写英文字母组成的字符串：“^[A-Z]+$”

只能输入由26个小写英文字母组成的字符串：“^[a-z]+$”

只能输入由数字和26个英文字母组成的字符串：“^[A-Za-z0-9]+$”

只能输入由数字、26个英文字母或者下划线组成的字符串：“^w+$”

验证用户密码:“^[a-zA-Z]w{5,17}$”正确格式为：以字母开头，长度在6-18之间，

只能包含字符、数字和下划线。

验证是否含有^%&’,;=?$”等字符：“[^%&’,;=?$x22]+”

只能输入汉字：“^[u4e00-u9fa5],{0,}$”

验证Email地址：“^w+[-+.]w+)\*@w+([-.]w+)\*.w+([-.]w+)\*$”

验证InternetURL：“^http://([w-]+.)+[w-]+(/[w-./?%&=]\*)?$”

验证电话号码：“^((d{3,4})|d{3,4}-)?d{7,8}$”

正确格式为：“XXXX-XXXXXXX”，“XXXX-XXXXXXXX”，“XXX-XXXXXXX”，

“XXX-XXXXXXXX”，“XXXXXXX”，“XXXXXXXX”。

验证身份证号（15位或18位数字）：“^d{15}|d{}18$”

验证一年的12个月：“^(0?[1-9]|1[0-2])$”正确格式为：“01”-“09”和“1”“12”

验证一个月的31天：“^((0?[1-9])|((1|2)[0-9])|30|31)$”

正确格式为：“01”“09”和“1”“31”。

匹配中文字符的正则表达式： [u4e00-u9fa5]

匹配双字节字符(包括汉字在内)：[^x00-xff]

匹配空行的正则表达式：n[s| ]\*r

匹配HTML标记的正则表达式：/< (.\*)>.\*|< (.\*) />/

匹配首尾空格的正则表达式：(^s\*)|(s\*$)

匹配Email地址的正则表达式：w+([-+.]w+)\*@w+([-.]w+)\*.w+([-.]w+)\*

匹配网址URL的正则表达式：http://([w-]+.)+[w-]+(/[w- ./?%&=]\*)?

## Java中判断字符串是否为数字的五种方法

1 //方法一：用JAVA自带的函数

2 public static boolean isNumeric(String str){

3 for (int i = str.length();--i>=0;){

4 if (!Character.isDigit(str.charAt(i))){

5 return false;

6 }

7 }

8 return true;

9 }

/\*方法二：推荐，速度最快

\* 判断是否为整数

\* @param str 传入的字符串

\* @return 是整数返回true,否则返回false

\*/

public static boolean isInteger(String str) {

Pattern pattern = Pattern.compile("^[-\\+]?[\\d]\*$");

return pattern.matcher(str).matches();

}

//方法三：

public static boolean isNumeric(String str){

Pattern pattern = Pattern.compile("[0-9]\*");

return pattern.matcher(str).matches();

}

//方法四：

public final static boolean isNumeric(String s) {

if (s != null && !"".equals(s.trim()))

return s.matches("^[0-9]\*$");

else

return false;

}

//方法五：用ascii码

public static boolean isNumeric(String str){

for(int i=str.length();--i>=0;){

int chr=str.charAt(i);

if(chr<48 || chr>57)

return false;

}

return true;

}

## HashMap按照键值key排序

Map map = new HashMap();

map.put("4", "a");

map.put("2", "b");

map.put("1", "c");

map.put("3", "d");

第一种方法：

Java代码 收藏代码

Object[] key = map.keySet().toArray();

Arrays.sort(key);

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

{

System.out.println(map.get(key[i]));

}

第二种方法：

Java代码 收藏代码

List arrayList = new ArrayList(map.entrySet());

Collections.sort(arrayList, new Comparator()

{

public int compare(Object arg1, Object arg2)

{

Map.Entry obj1 = (Map.Entry) arg1;

Map.Entry obj2 = (Map.Entry) arg2;

return (obj1.getKey()).toString().compareTo(obj2.getKey());

}

});

//将HASHMAP中的数据排序

for (Iterator iter = arrayList.iterator(); iter.hasNext();)

{

Map.Entry entry = (Map.Entry)iter.next();

String key = (String)entry.getKey();

System.out.println(map.get(key));

}

## java 基本数据类型的取值

一、

1、

基本类型：short 二进制位数：16

包装类：java.lang.Short

最小值：Short.MIN\_VALUE=-32768 （-2的15此方）

最大值：Short.MAX\_VALUE=32767 （2的15次方-1）

2、

基本类型：int 二进制位数：32

包装类：java.lang.Integer

最小值：Integer.MIN\_VALUE= -2147483648 （-2的31次方）

最大值：Integer.MAX\_VALUE= 2147483647 （2的31次方-1）

3、

基本类型：long 二进制位数：64

包装类：java.lang.Long

最小值：Long.MIN\_VALUE=-9223372036854775808 （-2的63次方）

最大值：Long.MAX\_VALUE=9223372036854775807 （2的63次方-1）

4、

基本类型：float 二进制位数：32

包装类：java.lang.Float

最小值：Float.MIN\_VALUE=1.4E-45 （2的-149次方）

最大值：Float.MAX\_VALUE=3.4028235E38 （2的128次方-1）

5、

基本类型：double 二进制位数：64

包装类：java.lang.Double

最小值：Double.MIN\_VALUE=4.9E-324 （2的-1074次方）

最大值：Double.MAX\_VALUE=1.7976931348623157E308 （2的1024次方-1）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **数据类型** | **字节数** | **二进制位数** | **范围** | **规律** |
| byte | 1 | 8 | -128～127 | -27～27-1 |
| short | 2 | 16 | -32768～32767 | -215～215-1 |
| int | 4 | 32 | -2147483648～2147483647 | -231～231-1 |
| long | 8 | 64 | -9223372036854775808 ~ 9223372036854775807 | -263～263-1 |
| float | 4 | 32 | 1.4E-45~3.4028235E38 |  |
| double | 8 | 64 | 4.9E-324~1.7976931348623157E308 |  |
| char | 2 | 16 | 0～65535 | 0~216-1 |
| boolean | 1 | 8 | true或false | true或false |

# IDE使用小技巧

## Eclipse编码方式

eclipse控制台的编码默认为操作系统的编码，比如GBK。这时候就不能正确显示utf8编码的汉字，需要指定控制台的编码为utf8。步骤：

菜单：Run Configuration | 右侧的选项卡Common 的 Console Encoding 选择UTF-8编码

## IDEA Method definition shorthands are not supported by current JavaScript version

问题：Method definition shorthands are not supported by current JavaScript version

表示：该方法定义的缺陷是不支持当前的JavaScript版本。

代码是没有问题的，但是出现红色波浪线

解决办法：

打开 File -> Settings -> Languages & Frameworks -> Javascript

更改JavaScript版本为ECMAScript 6

## Intellij IDEA运行报Command line is too long解法

报错内容:

Error running 'ServiceStarter': Command line is too long. Shorten command line for ServiceStarter or also for Application default configuration.

解法:

修改项目下 .idea\workspace.xml，找到标签 <component name="PropertiesComponent"> ， 在标签里加一行 <property name="dynamic.classpath" value="true" />

## 如何显示IntelliJ IDEA工具的Run Dashboard功能（靠谱）

How to enable the Spring Boot ‘Run Dashboard’ in IntelliJ IDEA 2017.2.1

从 JetBrains released IntelliJ IDEA 2017.2.1 版本之后，新出的功能‘Run Dashboard，它能非常方便的提供开发人员查看本地springboot服务运行的一些参数，功能相当优秀，但IDEA默认不显示此功能，你可以通过下面两种方式来打开它。

方法一：

通过编辑自定义VM选项并重新启动来实现！具体步骤如下：

1、在Help---Edit Custom VM Options中添加了 -Dide.run.dashboard = true 键值对；并重新启动IntelliJ IDEA！

2、在springboot项目的右上角运行配置中打开 Edit Configurations 界面，并在 Spring Boot Settings选项卡中，将Show in Run Dashboard 设为可用（选中即可）

现在就可以使你的运行在Run Dashboard中了，具体流程如下图：



方法二：

点击IDEA上面工具栏View---Tool Windows---Run Dashboard 即可打开界面，

如果Tool Windows的子选项里没有Run Dashboard，可以在工程目录下找.idea文件夹下的workspace.xml，在其中增加如下组件

<component name="RunDashboard">

<option name="configurationTypes">

<set>

<option value="SpringBootApplicationConfigurationType" />

</set>

</option>

<option name="ruleStates">

<list>

<RuleState>

<option name="name" value="ConfigurationTypeDashboardGroupingRule" />

</RuleState>

<RuleState>

<option name="name" value="StatusDashboardGroupingRule" />

</RuleState>

</list>

</option>

</component>

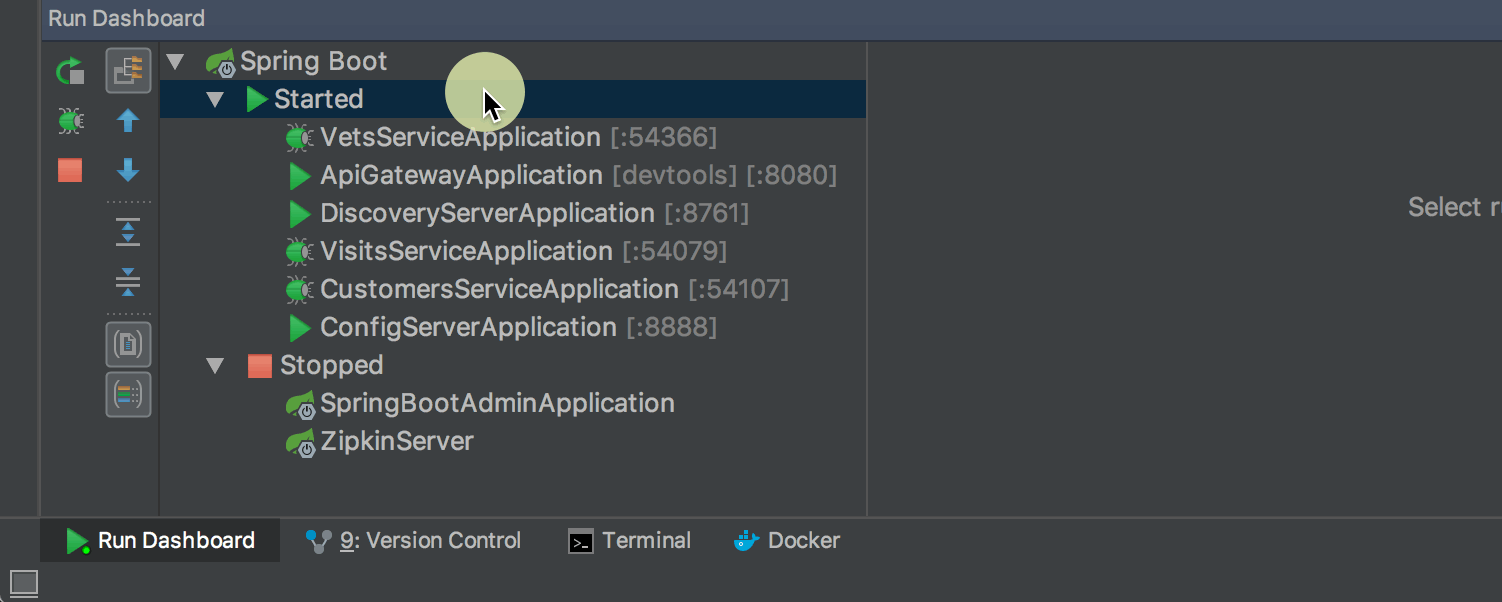
下面附录仪表盘使用的方法：

Spring Boot：仪表板和执行器端点

对于Spring Boot应用程序，传统的“运行”工具窗口已替换为“运行仪表板”。通过此仪表板，您可以管理运行配置并在右侧查看其特定于应用程序的信息：



此外，“ 运行 / 调试”工具窗口现在有一个名为“ 端点”的选项卡。此选项卡显示有关信息健康和豆类端点：

重要信息：因为它仍在进行中，您需要通过VM属性-Dide.run.dashboard = true或通过注册表显式启用它

# Office小技巧

## excel单元格设置为文本为何还要双击

excel单元格设置为文本时为何还要双击才有效？我选定一行，用右键设置为文本，为何还要单个逐一双击才有效，否则选定求和时（包含其中一个单元格），仍视为数值。而逐一双击太麻烦

一、确实是这样的，如果先在单元格中写入了数字，再设置单元格格式为文本格式时，是要双击进入单元格后回车才能让单元格中的数字转成文本形式存储的数字，这是EXCEL的一个规则，至于为什么这样只能问最初的开发设计人员了

二、如果是先设置单元格格式为文本再写入数字就不会出现这样的情况

三、如果你不想用双击这样的方法（再或是一整列的数字要转成文本数字），还可以用另一个方法进行设置，选中这个单元格(或这一整列单元格)---数据---分列---下一步--到步骤三时---列数据格式---选择"文本"---完成