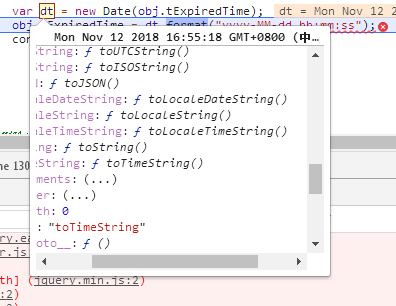
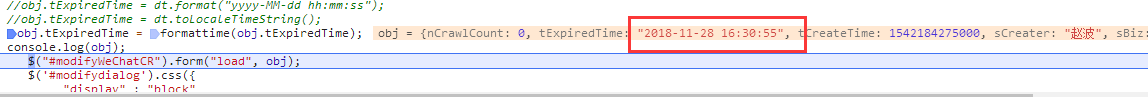
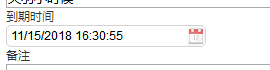


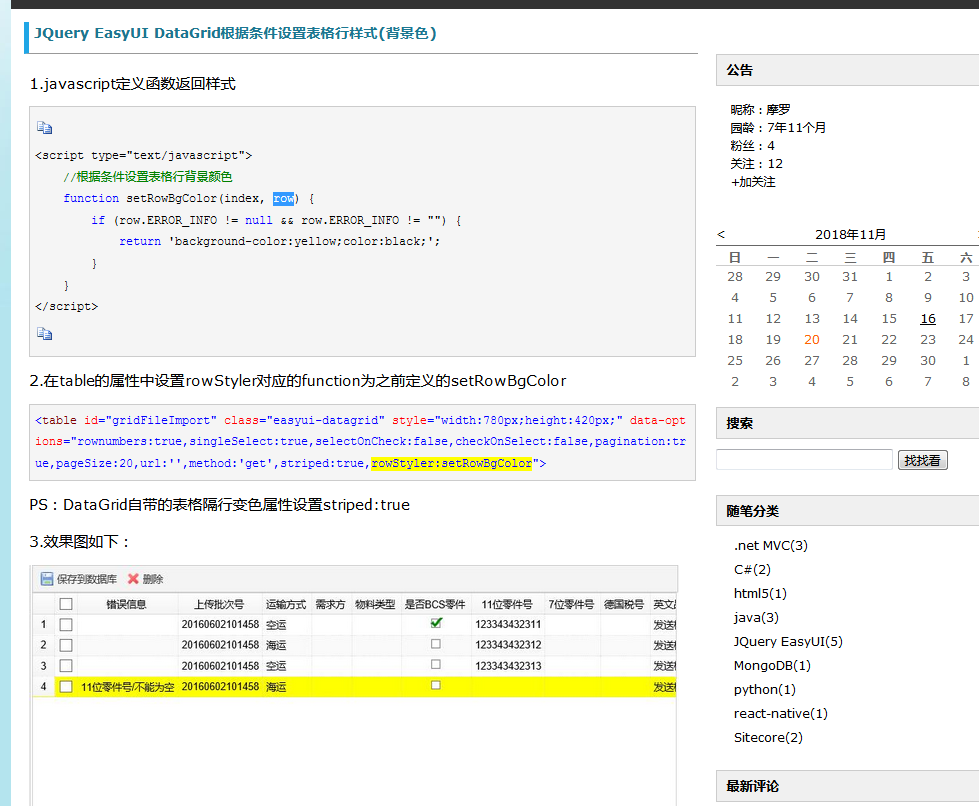


# 在js代码中使用断点查看一对象，可以看到该对象有哪些方法，这些方法都可以供我们在js中去调用

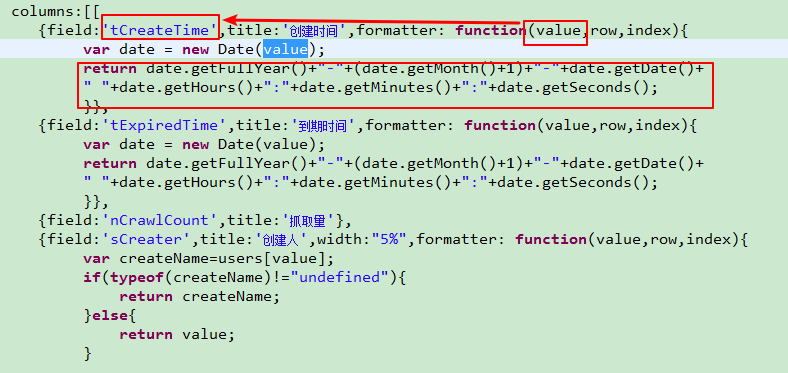




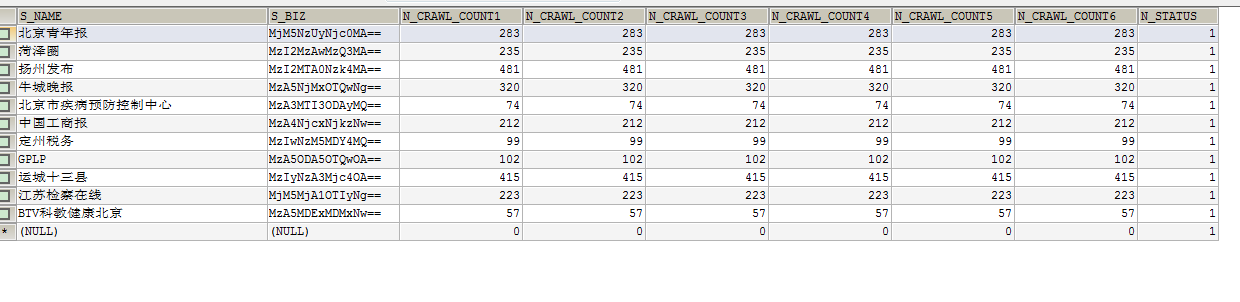


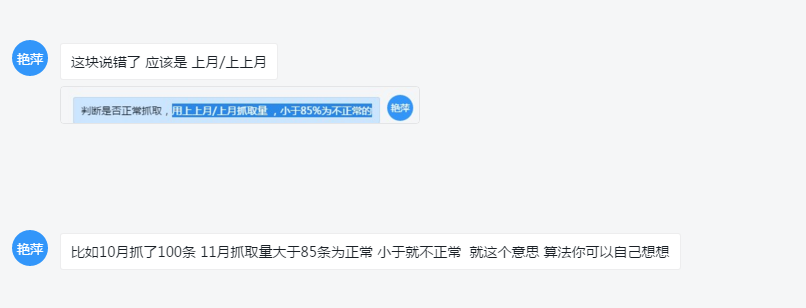


# easyUI中数据表格datagrid对从数据库中查询出来的数据是时间格式的数据处理











## mysql出现错误“ Every derived table must have its own alias”

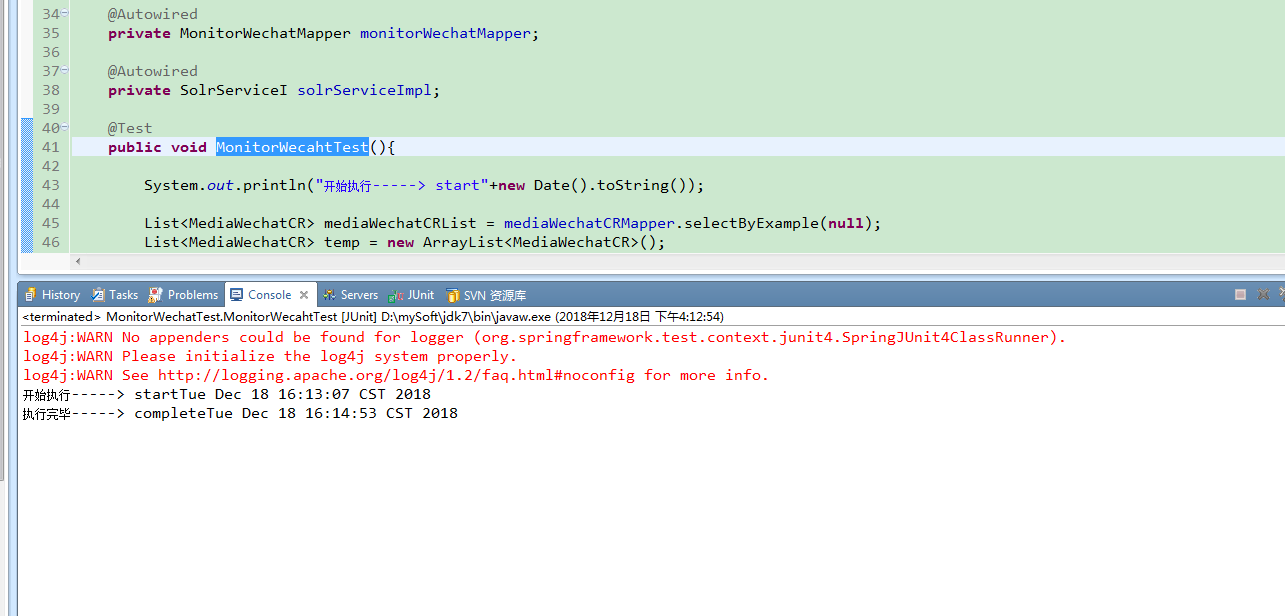


# Mysql中两个日期类型的字段相加减的查询语句https://www.aliyun.com/jiaocheng/1129812.html



# Java中对list集合进行分批处理





# [NameValuePair方式传参数](https://www.cnblogs.com/hunt/p/7071053.html)

www.cnblogs.com/hunt/p/7071053.html



package com.souche.lease.finance.product;

import com.alibaba.fastjson.JSON;

import com.alibaba.fastjson.JSONObject;

import org.apache.http.HttpStatus;

import org.apache.http.NameValuePair;

import org.apache.http.client.entity.UrlEncodedFormEntity;

import org.apache.http.client.methods.CloseableHttpResponse;

import org.apache.http.client.methods.HttpPost;

import org.apache.http.impl.client.CloseableHttpClient;

import org.apache.http.impl.client.HttpClients;

import org.apache.http.message.BasicNameValuePair;

import org.apache.http.protocol.HTTP;

import org.apache.http.util.EntityUtils;

import java.io.IOException;

import java.io.UnsupportedEncodingException;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

/\*\*

\* Created by hunt on 2017/6/26.

\* 使用HttpClient发送请求、接收响应很简单，一般需要如下几步即可。

\* 1. 创建HttpClient对象。

\* 2. 创建请求方法的实例，并指定请求URL。如果需要发送GET请求，创建HttpGet对象；如果需要发送POST请求，创建HttpPost对象。

\* 3. 如果需要发送请求参数，可调用HttpGet、HttpPost共同的setParams(HttpParams params)方法来添加请求参数；对于HttpPost对象而言，也可调用setEntity(HttpEntity entity)方法来设置请求参数。

\* 4. 调用HttpClient对象的execute(HttpUriRequest request)发送请求，该方法返回一个HttpResponse。

\* 5. 调用HttpResponse的getAllHeaders()、getHeaders(String name)等方法可获取服务器的响应头；调用HttpResponse的getEntity()方法可获取HttpEntity对象，该对象包装了服务器的响应内容。程序可通过该对象获取服务器的响应内容。

\* 6. 释放连接。无论执行方法是否成功，都必须释放连接

\*/

public class SDTestDemo {

public static void main(String[] args) {

String licenseNo = "浙A588AX";

String token = "7cc2bd72eb1e4522804dca3b88e8644d";

String city = "330100";

String timestamp = Long.toString(System.currentTimeMillis());

String sign;

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

mapParam.put("licenseNo", licenseNo);

mapParam.put("token", token);

mapParam.put("city", city);

mapParam.put("timestamp", timestamp);

SDTestUtil testUtil = new SDTestUtil();

sign = testUtil.MD5(testUtil.sort(mapParam));

mapParam.put("sign", sign);

/\*\*

\* 定义了一个list，该list的数据类型是NameValuePair（简单名称值对节点类型），

\* 这个代码用于Java像url发送Post请求。在发送post请求时用该list来存放参数。

\*/

List<NameValuePair> urlParameters = new ArrayList<>();

urlParameters.add(new BasicNameValuePair("licenseNo", licenseNo));

urlParameters.add(new BasicNameValuePair("token", token));

urlParameters.add(new BasicNameValuePair("city", city));

urlParameters.add(new BasicNameValuePair("timestamp", timestamp));

urlParameters.add(new BasicNameValuePair("sign", sign));

CloseableHttpClient httpclient = HttpClients.createDefault();

CloseableHttpResponse response = null;

HttpPost post = new HttpPost("http://101.231.154.154:8047/v4.0/renewal");

try {

post.setEntity(new UrlEncodedFormEntity(urlParameters, HTTP.UTF\_8));

try {

response = httpclient.execute(post);

// 判断网络连接状态码是否正常(0--200都数正常)

if (response.getStatusLine().getStatusCode() == HttpStatus.SC\_OK) {

String content = EntityUtils.toString(response.getEntity(), "UTF-8");

JSONObject jsonObject = JSON.parseObject(content);

System.out.println("报价返回内容是：" + jsonObject.toString());

if ("SUCCESS".equals(jsonObject.getString("code"))) {

System.out.println("成功，系统处理正常");

}

}

EntityUtils.consume(response.getEntity());//完全消耗

} catch (IOException e) {

e.printStackTrace();

} finally {

try {

if (null != response) response.close();

} catch (IOException e) {

e.printStackTrace();

}

}

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

} finally {

//释放链接

try {

httpclient.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

# Gson使用入门

http://www.cnblogs.com/majay/p/6336918.html



## Gson的创建方式一：直接new Gson对象

// 使用new方法

Gson gson = new Gson();

// toJson 将bean对象转换为json字符串

String jsonStr = gson.toJson(user, User.class);

// fromJson 将json字符串转为bean对象

Student user= gson.fromJson(jsonStr, User.class);

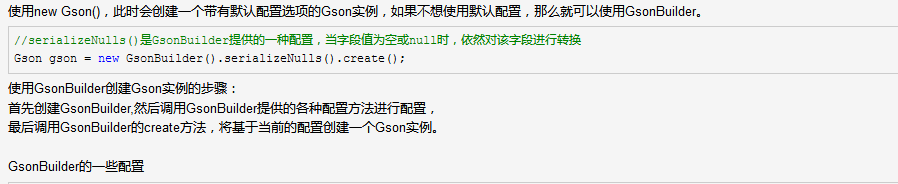
// \*\*序列化List\*\*： 将java对象转为json字符串

String jsonStr2 = gson.toJson(list);

// \*\*反序列化成List时需要使用到TypeToken getType()\*\*：json字符串转为java对象

List<User> retList = gson.fromJson(jsonStr2,new TypeToken<List<User>>(){}.getType());

## Gson的创建方式二：使用GsonBuilder



Gson gson = new GsonBuilder()

.excludeFieldsWithoutExposeAnnotation() //不对没有用@Expose注解的属性进行操作

.enableComplexMapKeySerialization() //当Map的key为复杂对象时,需要开启该方法

.serializeNulls() //当字段值为空或null时，依然对该字段进行转换

.setDateFormat("yyyy-MM-dd HH:mm:ss:SSS") //时间转化为特定格式

.setPrettyPrinting() //对结果进行格式化，增加换行

.disableHtmlEscaping() //防止特殊字符出现乱码

.registerTypeAdapter(User.class,new UserAdapter()) //为某特定对象设置固定的序列或反序列方式，自定义Adapter需实现JsonSerializer或者JsonDeserializer接口

.create();



Gson gson = new GsonBuilder().enableComplexMapKeySerialization().create(); //开启复杂处理Map方法

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

// TODO 向map中添加数据

String jsonStr = gson.toJson(map); //toJson

Map<List<User>, String> resultMap = gson.fromJson(jsonStr,new TypeToken<Map<List<User>, String>>() {}.getType()); //fromJson

## Gson全解析<https://www.jianshu.com/p/fc5c9cdf3aab>

# Spring中bean工厂后置处理器的使用

## 后置处理器（BeanFactoryPostProcessor）

https://www.jianshu.com/p/b45efc018bcc

[SpringIOC源码阅读—BeanDefinitionDocumentReader](https://www.jianshu.com/p/339c87aae174)我们了解到Spring对bean定义的载入有很多种方式，读取的过程是可插拔的，不论何种形式，spring的IOC容器只要获得了bean定义信息，都可以正常工作。而我们熟知的配置读取方式就是XML文件，**如果你希望，可以自己定制配置信息的读取过程，这就是Spring的特性体现之一。**  
同样BeanFactoryPostProcessor也是Spring可扩展性的一个体现，

**允许我们在工厂里所有的bean被加载进来后但是还没初始化前，对所有bean的属性进行修改也可以add属性值。**

## [PropertyPlaceholderConfigurer读取配置文件](https://www.cnblogs.com/dream-to-pku/p/6367396.html)

http://www.cnblogs.com/dream-to-pku/p/6367396.html

1. PropertyPlaceholderConfigurer是个bean工厂后置处理器的实现，也就是 BeanFactoryPostProcessor接口的一个实现。PropertyPlaceholderConfigurer可以将上下文（配置文 件）中的属性值放在另一个单独的标准java Properties文件中去。在XML文件中用${key}替换指定的properties文件中的值。这样的话，只需要对properties文件进 行修改，而不用对xml配置文件进行修改。

2.在Spring中，使用PropertyPlaceholderConfigurer可以在XML配置文件中加入外部属性文件，当然也可以指定外部文件的编码，如：

<bean id="propertyConfigurer"class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

<property name="location">

<value>conf/jdbc.properties</value>

</property>

<property name="fileEncoding">

<value>UTF-8</value>

</property>

</bean>

3.PropertyPlaceholderConfigurer起的作用就是将占位符指向的数据库配置信息放在bean中定义的工具。

4.查看源代码，可以发现，locations属性定义在PropertyPlaceholderConfigurer的祖父类 PropertiesLoaderSupport中，而location只有 setter方法。类似于这样的配置，在spring的源程序中很常见的。

PropertyPlaceholderConfigurer如果在指定的Properties文件中找不到你想使用的属性，它还会在Java的System类属性中查找。

我们可以通过System.setProperty(key, value)或者java中通过-Dnamevalue来给Spring配置文件传递参数。

<bean class="com.slp.util.SpringContextHolder"/>

<bean id="propertyConfigurer"

class="com.slp.util.CustomizedPropertyConfigurer">

<property name="locations">

<list>

<value>classpath:conf/jdbc.properties</value>

<value>classpath:conf/config.properties</value>

<value>classpath:conf/server.properties</value>

</list>

</property>

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

</bean>

使用代码示例：

package com.slp.util;

import org.springframework.beans.BeansException;

import org.springframework.beans.factory.config.ConfigurableListableBeanFactory;

import org.springframework.beans.factory.config.PropertyPlaceholderConfigurer;

import java.util.HashMap;

import java.util.Map;

import java.util.Properties;

public class CustomizedPropertyConfigurer extends PropertyPlaceholderConfigurer {

private static Map<String,String> ctxPropMap;

@Override

protected void processProperties(ConfigurableListableBeanFactory beanFactoryToProcess, Properties props) throws BeansException {

super.processProperties(beanFactoryToProcess, props);

ctxPropMap = new HashMap<>();

for (Object key : props.keySet()){

String keyStr = key.toString();

String value = String.valueOf(props.get(keyStr));

ctxPropMap.put(keyStr,value);

}

}

public static String getCtxProp(String name) {

return ctxPropMap.get(name);

}

public static Map<String, String> getCtxPropMap() {

return ctxPropMap;

}

}

# mybatis Examplmybatis Example Criteria like 模糊查询e Criteria like 模糊查询

XXXExample类用于构造复杂的筛选条件

它包含一个名为Criteria的内部静态类，它包含将在where子句中一起结合的条件列表。

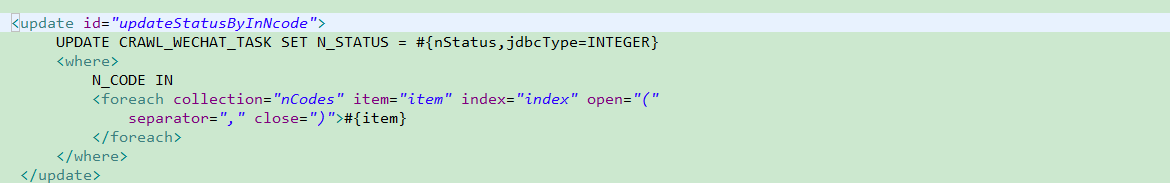
Criteria类的集合允许您生成几乎无限类型的where子句。

可以使用createCriteria方法或or方法创建Criteria对象。

当使用createCriteria方法创建第一个Criteria对象时，它会自动添加到Criteria对象列表中 -

如果不需要其他子句，则可以轻松编写简单的Where子句。使用or方法时，Criteria类将添加到所有实例的列表中。





# Spring-boot 整合

<https://blog.csdn.net/qq_40369944/column/info/29676>

IDEA注册码：https://blog.csdn.net/zixiao217/article/details/82942476



# Spring-cloud入门

https://blog.csdn.net/qq\_40369944/column/info/31310



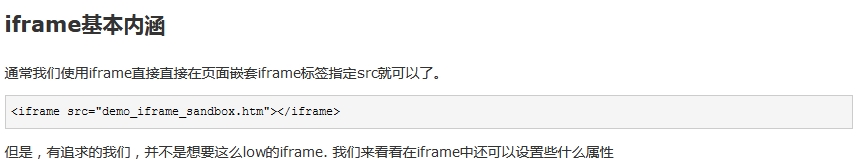
# LayUI入门

<https://www.layui.com/doc/element/layout.html>



# [Web前端之iframe详解](https://www.cnblogs.com/lvhw/p/7107436.html)

<http://www.cnblogs.com/lvhw/p/7107436.html>



iframe常用属性:

1.frameborder:是否显示边框，1(yes),0(no)

2.height:框架作为一个普通元素的高度，建议在使用css设置。

3.width:框架作为一个普通元素的宽度，建议使用css设置。

4.name:框架的名称，window.frames[name]时专用的属性。

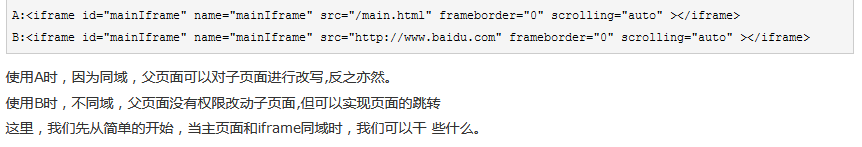
5.scrolling:框架的是否滚动。yes,no,auto。

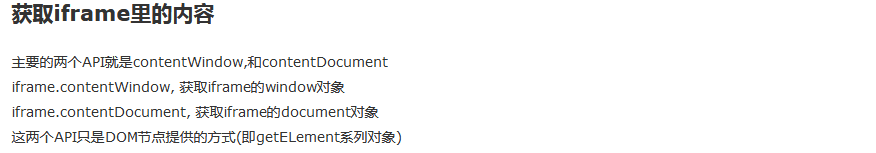
6.src：内框架的地址，可以使页面地址，也可以是图片的地址。

7.srcdoc , 用来替代原来HTML body里面的内容。但是IE不支持, 不过也没什么卵用

8.sandbox: 对iframe进行一些列限制，IE10+支持

上面一些tag，会在下文进行穿插说明，单个不好说。  
我们通常使用iframe最基本的特性，就是能自由操作iframe和父框架的内容(DOM). 但前提条件是同域. 如果跨域顶多只能实现页面跳转window.location.href.  
那什么是同域/ 什么是跨域呢?  
就是判断你的url首部是否一样，下面会有讲解，这里只是提及。  
同域不同域的问题:





var iframe = document.getElementById("iframe1");

var iwindow = iframe.contentWindow;

var idoc = iwindow.document;

console.log("window",iwindow);//获取iframe的window对象

console.log("document",idoc); //获取iframe的document

console.log("html",idoc.documentElement);//获取iframe的html

console.log("head",idoc.head); //获取head

console.log("body",idoc.body); //获取body

# maven配置教程

<https://www.cnblogs.com/hero123/p/9092581.html>

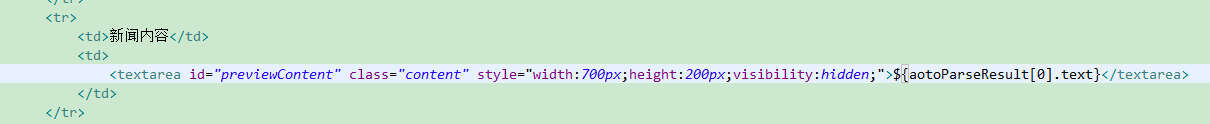


# 相对路径./和../的区别

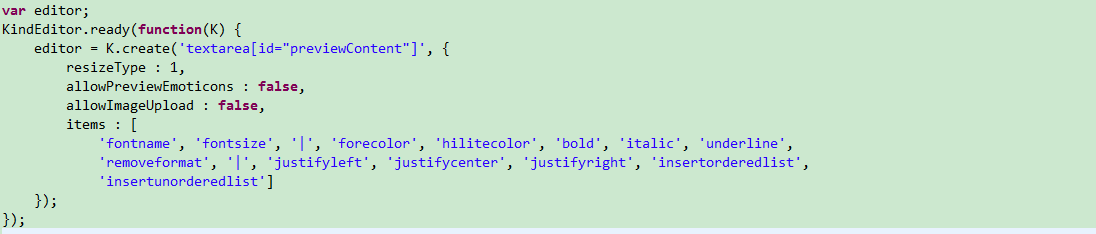
<https://blog.csdn.net/qq_34769573/article/details/80445681>

# 富文本编辑器的使用

1. 页面代码



1. JS代码（类似先定义一个富文本编辑器对象）



**var** editor;

KindEditor.ready(**function**(K) {

editor = K.create('textarea[id="previewContent"]', {

resizeType : 1,

allowPreviewEmoticons : **false**,

allowImageUpload : **false**,

items : [

'fontname', 'fontsize', '|', 'forecolor', 'hilitecolor', 'bold', 'italic', 'underline',

'removeformat', '|', 'justifyleft', 'justifycenter', 'justifyright', 'insertorderedlist',

'insertunorderedlist']

});

});

1. 通过定义的富文本编辑器对象，获取富文本编辑器中的内容

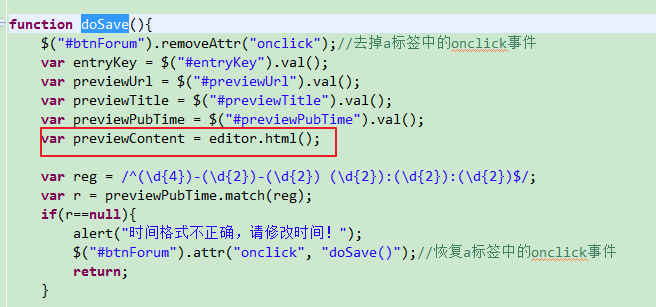
**var** entryKey = $("#entryKey").val();

**var** previewUrl = $("#previewUrl").val();

**var** previewTitle = $("#previewTitle").val();

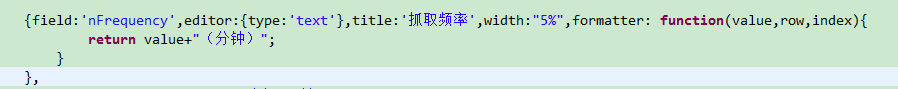
**var** previewPubTime = $("#previewPubTime").val();

**var** previewContent = editor.html();

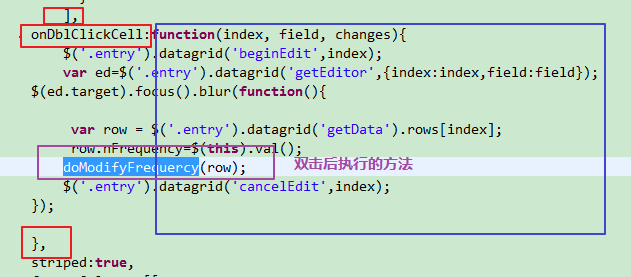


# EasyUI 双击修改单元格代码示例

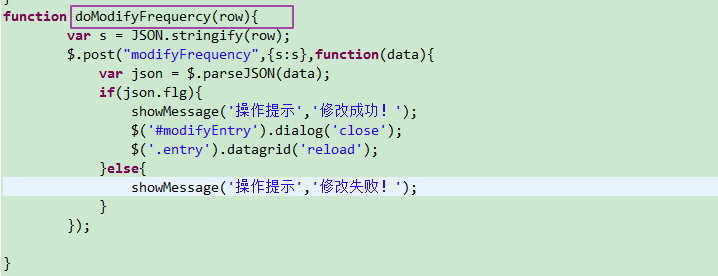
1. 首先在columns列属性中定义editor 属性（详见参考手册）



1. 在Data Grid中使用onDblClickCell 属性来指定双击（单机雷同详见参考手册）



1. 修改后执行的方法就是一个保存操作



# JsonObject 的getString方法

1.简单的解析json字符串首先将json字符串转换为json对象，然后再解析json对象，过程如下。

JSONObject jsonObject = JSONObject.fromObject(jsonStr);

根据json中的键得到它的值String name = jsonObject.getString("name");

int numb = jsonObject.getInt("num");

String sex = jsonObject.getString ("sex");

int age = jsonObject.getInt ("age");

2.将json字符串转换为java对象同样先将json字符串转换为json对象，再将json对象转换为java对象，如下所示。

JSONObject obj = new JSONObject().fromObject(jsonStr);//将json字符串转换为json对象

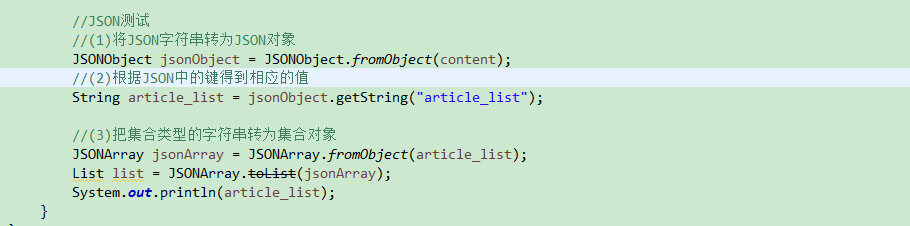
将json对象转换为java对象Person jb = (Person)JSONObject.toBean(obj,Person.class);//将建json对象转换为Person对象

3.将java对象转换为json字符串先将java对象转换为json对象，在将json对象转换为json字符串JSONObject json = JSONObject.fromObject(obj);//将java对象转换为json对象String str = json.toString();//将json对象转换为字符串

JSONObject jsonObject = JSONObject.*fromObject*(content);

JSONArray jsonArray = JSONArray.fromObject(article\_list);

以上两个方法都是将JSON字符串转为JSON对象，一个是普通的json对象，一个是想转为集合的json对象



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

作者：MakeGreatEffort

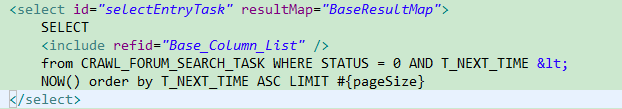
来源：CSDN

原文：https://blog.csdn.net/qq\_36838191/article/details/79867655

版权声明：本文为博主原创文章，转载请附上博文链接！

# Mybatis 中的大于小于号





# IO流一次性读取文件中的所有内容

https://blog.csdn.net/qq\_28483283/article/details/81128955



# 利用IO流下载文件



# Java中的线程池

https://www.cnblogs.com/exe19/p/5359885.html

