


Layui Table 使用特定的方法返回数据

Table 的js 实现

```
layer = layui.layer;
layui.use('table', function() {
    table = layui.table;
    form = layui.form;
    table.render({
        elem: '#tableId' // 对应的id
        // ,url: urls + '?semailids='+semailids
        ,url: '/yzcr/subordinate/findSubordinateMailALLAjax?semailids='+semailids // 接口的url
        ,skin: 'line' // 表格样式
        ,cols: [[ // 表格的列
            {type: 'checkbox', width: "2%"}
            ,{field: 'id', width: "2%", title: '邮件id', style: "display: hidden"}
            ,{field: 'emid', width: "2%", title: '邮箱id', style: "display: hidden"}
            ,{field: 'type', width: "2%", title: '邮件类型', style: "display: hidden"}
            ,{field: 'nickname', width: "8%", title: '下属昵称'}
            ,{field: 'mail_from', width: "10%", title: '昵称/地址'}
            ,{field: 'subject', width: "20%", event: 'setSign', templet: '#subject', title: '主题'}
            ,{field: 'body_text', width: "30%", templet: '#label', templet: '#bodyText', title: '邮件正文'}
            ,{field: 'has_attachment', width: "5%", templet: '#attachment', title: '附件'}
            // ,{field: 'mail_from', width: "6%", title: '分发'}
            // ,{field: 'mail_from', width: "6%", title: '删除'}
            ,{field: 'sent_date', width: "10%", title: '时间'}
        ]]
        // 对应返回数据的字段
        ,page: true // 开启分页
    });
});
```

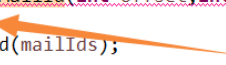
充当id, <script id="attachment"> 这样引用

```
<!-- 附件状态 -->
<script type="text/html" id="attachment">
{{# if(d.hasAttachment === 'false'){ }}
    否
{{#   } else { }}
    是
{{#   } }}
</script>
```



Mybatis PageHelper 插件的分页统计

```
@Override
public List<Map<String, Object>> messageByMailId(int offset, int limit, String mailIds) {
    PageHelper.offsetPage(offset, limit);
    return sysMessageMapper.messageByMailId(mailIds);
}
```



Mybatis 使用 IN 关键字时，用\${}来获取值，不需要用#{ }来进行占位（预编译）

Thymeleaf 在前台取出后台存到 request | session | application 中的值
以`[[${variable}]]` | `[[${session.variable}]]`的形式取后台传过来的参数

Layui laytpl 模板引擎

其中的 d 表示数据(data)

```
findSubordinateMailALL.js
149 //      ,{field:'id', style: 'display: none'} // 邮件id
150 //      ,{field:'emid', style: 'display: none'} // 邮箱id
151 //      ,{field:'type', style: 'display: none'} // 邮件类型, 1 发件 2 收件
152      ,{field:'nickname',width:"8%", title: '下属昵称'}
153      ,{field:'mail_from',width:"10%", title: '昵称/地址'}
154      ,{field:'subject', width:"20%", event: 'setSign',templet: '#subject', title: '主题'}
155      ,{field:'body_text',width:"30%", templet: '#bodyText', title: '邮件正文'}
156      ,{field:'has_attachment', width:"5%",templet: '#attachment', title: '附件'}
157      ,{field:'', width:"6%", title: '分发'}
158      ,{field:'', width:"6%", title: '删除'}
159      ,{field:'sent_date', width:"10%",title: '时间'}
160

SubordinateMailController.java  *findSubordinateMailALL.html  SubordinateMailService.java  SubordinateMailServiceImp.java  SubordinateMailMapper.xml
139 </script>
140
141 <!-- 附件状态 -->
142 <script type="text/html" id="attachment">
143     {{# if(d.hasAttachment === 'false'){ }}
144     否
145     {{# } else { }}
146     是
147     {{# } }}
148 </script>
149
150 <!-- 未读标记 -->
```

这个id要和layui中的对应。方便格式化

d表示一个表格的每行数据，一行数据就是一个对象，简称对象 d

Layui 表单渲染

```
form.render('select'); layui的 from表单 使用form.render重载
```

```
table.reload('tableId', { layui的 table数据表格 使用table.reload重载  
    url: '/yzcr/subordinate/findSubordinateMailALLAjax?semailids=' + xiamail  
});
```

Thymeleaf 中的

th:selected、th:if、th:value、th:onclick 标签实践，th:onclick：作用同 html 的 onclick。

th:include 标签与 html 标签中的 include 意义相同（包含另一个页面），不过要与 th:fragment 标签结合使用

1. th:insert:保留自己的主标签，保留th:fragment的主标签。
2. th:replace:不要自己的主标签，保留th:fragment的主标签。
3. th:include:保留自己的主标签，不要th:fragment的主标签。（官方3.0后不推荐）

```
<div th:include="footer :: copy"></div> th:fragment="copy"  
    模板名，可以是html名
```

百度的富文本编辑器 Ueditor

ueditor 的 serverUrl

- `serverUrl` {Path String} [默认值：URL + "php/controller.php"] // 服务器统一请求接口路径

例如图片的上传，那个接口就可以写后台的图片上传接口。

jQuery

`$("input[type='text'][name='subject']").val(tname);`

`$("input[type='text'],[name='subject']").val(tname);`

筛选 input 标签，前者是与条件，后者或条件

Layui 自定义模板


数据表格中使用 id 然后在 html 中进行格式话，或者直接操作，后边要 return 出去

```
<!-- ssl状态 -->
<script type="text/html" id="sslstate">
{{# if(d.isSsl === 'false'){ }}
    否
    {{#   } else { }}
    是
    {{#   } }}
</script>
```

id要和templet的值对应

这是layui的模板引擎写法, 例如{{# }}

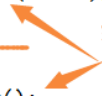
```
,{field:'emailList', width:'45%', align:'center', title: '邮箱',  
  templet: function(data) {  
    if (data.emailList.length == 0) {  
      return "";  
    }  
    var html = "<select lay-ignore id='se" + i + "'>";  
    layui.each(data.emailList, function(index, item) {  
      if (index == 0) {  
        html += '<option selected value=' + item.id + '>' + item.account + '</option>';  
      } else {  
        html += '<option value=' + item.id + '>' + item.account + '</option>';  
      }  
    });  
    html += '</select>';  
    return html;  
  }}  
}
```



jQuery

使用jquery 自定义事件然后自动触发


```
$("#close").click(function() {  
    layer.load(1);  
    var index = parent.layer.getFrameIndex(window.name); //先得到当前iframe层的索引  
    parent.layer.close(index); //再执行关闭  
});  
$(function() {  
    $("#close").click();  
})
```



给该id绑定一个事件

然后自动触发

[\\$\(\).each](#) && [\\$.each](#)

前者更多是用于处理 DOM 对象，如遍历某 div 下所有的 span；

后者更多是用于处理数据，如遍历数组、对象。

jQuery

jQuery 的 next()，表示该元素的紧邻的下一个同辈元素

```

<title>架构教程(runoob.com)</title>
<style>
.siblings *{
  display: block;
  border: 2px solid lightgrey;
  color: lightgrey;
  padding: 5px;
  margin: 15px;
}
</style>
<script src="https://cdn.bootcss.com/jquery/1.10.2/jquery.min.js">
</script>
<script>
$(document).ready(function(){
  $(".li.start").next().css({"color":"red","border":"2px solid red"});
});
</script>
</head>
<body>

<div style="width:500px;" class="siblings">
  <ul>ul (父节点)
    <li>li (兄弟节点)</li>
    <li>li (兄弟节点)</li>
    <li class="start">li (类名为"start"的兄弟节点)</li>
    <li>li (类名为"start"的li节点的下一个的兄弟节点)</li>
    <li>li (兄弟节点)</li>
  </ul>
</div>

</body>
</html>

```



概述

查找当前元素之后所有的同辈元素。

可以用表达式过滤

nextAll()

```

.siblings {
  display: block;
  border: 2px solid lightgrey;
  color: lightgrey;
  padding: 5px;
  margin: 15px;
}
</style>
<script src="https://cdn.bootcss.com/jquery/1.10.2/jquery.min.js">
</script>
<script>
$(document).ready(function(){
  $(".li.start").nextAll().css({"color":"red","border":"2px solid red"});
});
</script>
</head>
<body>

<div style="width:500px;" class="siblings">
  <ul>ul (父节点)
    <li>li (兄弟节点)</li>
    <li>li (兄弟节点)</li>
    <li class="start">li (类名为"star"的兄弟节点)</li>
    <li>li (类名为"star"的li节点的下一个兄弟节点)</li>
    <li>li (类名为"star"的li节点的下一个兄弟节点)</li>
    <li>li (类名为"star"的li节点的下一个兄弟节点)</li>
  </ul>
</div>
<p>在这个例子中,我们将返回类名为"star"的li节点的所有下一个兄弟节点。</p>

</body>
</html>

```



在这个例子中,我们将返回类名为 "star" 的li节点的所有下一个兄弟节点

find() 用于查找元素

描述:

从所有的段落开始，进一步搜索下面的span元素。与\$("p span")相同。

HTML 代码:

```
<p><span>Hello</span>, how are you?</p>
```

jQuery 代码:

```
$("p").find("span")
```

结果:

```
[ <span>Hello</span> ]
```

Layui 表格重载，table.reload(filterId);

Spring 注解

@Value("\${}")，用于获取 properties 属性文件的属性

@Value("#{}")，用于获取类的属性。初步认识

@Resource，默认根据 byName 进行注入，找不到再根据类型注入

@Autowired，根据类型 byType 注入

Jquery 的 removeClass()方法，移除类名，

Layui 的 layue.closeAll();关闭所有弹窗(layer.open())

mybatis 的 Example 类及一些常用方法。

使用这个类，可以搭建无限的 where 条件。

一般可以使用，and() 多个条件 AND 连接、or()多个条件 OR 连接

•排序 example.setOrderByClause("sent_date desc")、

已有的方法

•andEqualTo("userId", userId)

•andIn("state", list)

•andLike("bodyText", word)


或者自定义条件

•example.createCriteria().andCondition("user_id in (" +userId+ ")");

•example.and().andCondition("user_id in (" +userId+ ")");

可以

```
Example example = new Example(SysGarbage.class);
example.setOrderByClause("id desc");
if(keyword=="1")
    example.createCriteria().andEqualTo("userid", getCurrentUserId()).andEqualTo("gtype", "1");
if(keyword=="2")
    example.createCriteria().andEqualTo("userid", getCurrentUserId()).andEqualTo("gtype", "2");
return sysGarbageMapper.selectByExample(example);
```



或者

```
Example example = new Example(SysEmail.class);
if(userId!=null&&userId!="") {
    example.createCriteria().andCondition("user_id in (" +userId+ ")");
    return sysEmailMapper.selectByExample(example);
} else {
    return null;
}
```

再或者

```
Example example=new Example(RegisterUser.class);
example.setOrderByClause(" id desc ");
return registerUserMapper.selectByExample(example);
```

Mapper 文件

```
import java.util.List;

import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Param;

import com.lf.sys.model.SysResource;
import com.lf.utils.MyMapper;
@Mapper
public interface SysResourceMapper extends MyMapper<SysResource> {

    /**界面列表查询*/
    List<SysResource> findRnameAndRestype(@Param("rname") String rname,@Param("resource_type") String resource_type);

    /**根据公司id查询对应资源*/
    List<SysResource> findOrgAndRes(@Param("orgId") String orgId);
}
```


jQuery 移除事件

[\\$\('#btn'\).unbind\("click"\);](#) //移除 click

[\\$\('#btn'\).unbind\(\);](#) //移除所有,3.0 版本中该方法废弃, 推荐使用 [off\(\)](#)

`unbind(type [,data])` //data 是要移除的函数

`$('#btn').one("click",function(){.....});` // 只触发一次的事件

Thymeleaf 的一个内置对象

strings 如使用到它来截取分隔的字符, 注, 使用内置对象要`${#}`这样的形式

`${#strings.arraySplit}`, 其中一个。分隔

```
/*
 * Split and join
 */
${#strings.arrayJoin(namesArray, ',')}
${#strings.listJoin(namesList, ',')}
${#strings.setJoin(namesSet, ',')}
${#strings.arraySplit(namesStr, ',')}           // returns String[]
${#strings.listSplit(namesStr, ',')}           // returns List<String>
${#strings.setSplit(namesStr, ',')}             // returns Set<String>
```

Jquery 的 siblings 用于选择所有的同胞元素

实例

查找每个 p 元素的所有类名为 "selected" 的所有同胞元素：

```
$("p").siblings(".selected")
```

Jquery 的 on()，绑定事件推荐使用这个，不推荐 bind()

on(events,[selector],[data],fn)

概述

在选择元素上绑定一个或多个事件的事件处理函数。

events,[selector],[data],fn

V1.7

events: 一个或多个用空格分隔的事件类型和可选的命名空间，如"click"或"keydown.myPlugin"。

selector: 一个选择器字符串用于过滤器的触发事件的选择器元素的后代。如果选择的< null或省略，当它到达选定的元素，事件总是触发。

data: 当一个事件被触发时要传递event.data给事件处理函数。

fn: 该事件被触发时执行的函数。 false 值也可以做一个函数的简写，返回false。

events-map,[selector],[data]

V1.7

events-map: 一个用字符串表示的，一个或多个空格分隔的事件类型和可选的命名空间，值表示事件绑定的处理函数。

selector: 一个选择器字符串过滤选定的元素，该选择器的后裔元素将调用处理程序。如果选择是空或被忽略，当它到达选定的元素，事件总是触发。

data: 当一个事件被触发时要传递event.data给事件处理函数。

Jquery 的 off()，用于卸载事件，不推荐 unbind()

off(events,[selector],[fn])

概述

在选择元素上移除一个或多个事件的事件处理函数。

参数

events,[selector],[fn]

V1.7

events: 一个或多个空格分隔的事件类型和可选的命名空间，或仅仅是命名空间，比如"click", "keydown.myPlugin", 或者 ".myPlugin".

selector: 一个最初传递到.on()事件处理程序附加的选择器。

fn: 事件处理程序函数以前附加事件上，或特殊值false.

events-map,[selector]

V1.7

events-map: 一个用字符串表示的，一个或多个空格分隔的事件类型和可选的命名空间，值表示先前事件绑定的处理函数。

selector: 一个最初传递到.on()事件处理程序附加的选择器。

Jquery prop()方法用于设置或返回被选元素的属性和值

返回属性的值：

```
$(selector).prop(property)
```

设置属性和值：

```
$(selector).prop(property,value)
```

使用函数设置属性和值：

```
$(selector).prop(property,function(index,currentvalue))
```

设置多个属性和值：

```
$(selector).prop({property:value, property:value,...})
```

prop()设置的属性应该由 removeProp()来进行移除

`removeProp()` 方法移除由 `prop()` 方法设置的属性。

注意：不要使用该方法来移除诸如 `style`、`id` 或 `checked` 之类的 HTML 属性。请使用 `removeAttr()` 方法代替。

例如

从所有的 `<p>` 元素移除样式属性：

```
$("#button").click(function(){
    $("p").removeAttr("style");
});
```

尝试一下 »

定义和用法

`removeAttr()` 方法从被选元素移除一个或多个属性。

语法

```
$(selector).removeAttr(attribute)
```

| 参数 | 描述 |
|------------------|---|
| <i>attribute</i> | 必需。规定要移除的一个或多个属性。如需移除若干个属性，请使用空格分隔属性名称。 |

Jquery 的 load 方法

load(url, [data], [callback])

概述

载入远程 HTML 文件代码并插入至 DOM 中。

默认使用 GET 方式 - 传递附加参数时自动转换为 POST 方式。jQuery 1.2

参数

url,[data],[callback]]

url:待装入 HTML 网页网址。

data:发送至服务器的 key/value 数据。在jQuery 1.3中也可以接受一个字符串

callback:载入成功时回调函数。

示例

把文件 "demo_test.txt" 的内容加载到指定的 <div> 元素：

```
$("#button").click(function(){  
    $("#div1").load("demo_test.txt");  
});
```

SpringMVC 框架中，controller 方法返回类型为 void，那么就会找和 url **名字相同的页面**

<http://www.iteye.com/problems/109836>

https://blog.csdn.net/weixin_38617682/article/details/78754602

默认调用 dispatcher 方法吗？

jquery 序列化追加数据，可以这样

```
$( "from-id" ).serialize() + "&parame=" + value;
```

[@Transient](#) 注解

表示该属性不是表中的字段，但是联表的时候可以用上。在类属性上进行注解，然后在 mapper 中也配置下，联表查询时可以接受该字段了。

Layui 弹窗 (open())

layui.open()时，弹出的 content 可以时 id 也可以时 class

```
content:$( "#id" ) || content:$( ".class" );
```

bootstrap table 更改样式通过 data-formatter 来进行

可以修改样式，如 data-formatter=" style()" ，js 中 function style()

jquery 键盘事件 keypress、keydown 和 keyup

指键盘按下-放开这个过程、键盘按下、键盘放开

keypress:

完整的 key press 过程分为两个部分：1. 按键被按下；2. 按键被松开。

当按钮被按下时，发生 keydown 事件。

keydown() 方法触发 keydown 事件，或规定当发生 keydown 事件时运行的函数。

注释：如果在文档元素上进行设置，则无论元素是否获得焦点，该事件都会发生。

keydown:

当按钮被按下时，会发生该事件。它发生在当前获得焦点的元素上。

keyup:

当按钮被松开时，发生 keyup 事件。它发生在当前获得焦点的元素上。

jQuery 的 each 跳出循环 ,

```
$.each (list, function(index, item) {  
    console.log( “跳出循环(等同于 break):return false;” );  
    console.log( “跳出当前循环(等同于 continue): return true;” );  
});
```

jQuery ajax 的 timeout 设置

可限制 ajax 的执行时间 , 到了超时时间自动执行 error 回调

Bootstrap 模态窗口关闭 , 可监听

模态窗口关闭时调用

```
$(function() {  
    // 当客户公海添加表单关闭时调用  
    $("#seasAddModal").on("hide.bs.modal", function() {  
        var seasstate2 = $("#seasstate2");  
        var seasstate2Parent = seasstate2.parent("label");  
        seasstate2.removeAttr("disabled");  
        seasstate2Parent.removeAttr("style");  
    });  
});
```

| | |
|-----------------|-------------------------------|
| show.bs.modal | 在调用 show 方法后触发。 |
| shown.bs.modal | 当模态框对用户可见时触发（将等待 CSS 过渡效果完成）。 |
| hide.bs.modal | 当调用 hide 实例方法时触发。 |
| hidden.bs.modal | 当模态框完全对用户隐藏时触发。 |

<https://blog.csdn.net/wn464319127/article/details/50432965>

Bootstrap 表格的行监听


```
$("#list_notices").on("click-row.bs.table", function(value, row, element)
    layer.open({
        title: "今日登录详情",
        type: 2,
        shadeClose: true,
        scrollbar: false,
        content: "loginhistory/loginDetail?userid=" + row.userid,
        area: ['750px', '460px']
    });
});
```

[jQuery.extend](#)

用于合并一个对象

target,[object1],[objectN]

Object,Object,Object

target: 一个对象，如果附加的对象被传递给这个方法将那么它将接收新的属性，如果它是唯一的参数将扩展jQuery的命名空间。

object1: 待合并到第一个对象的对象。

objectN: 待合并到第一个对象的对象。

[deep],target,object1,[objectN]

Object,Object,Object,Object

deep: 如果设为true，则递归合并。

target: 待修改对象。

object1: 待合并到第一个对象的对象。

objectN: 待合并到第一个对象的对象。

描述:

合并 settings 和 options , 修改并返回 settings。

jQuery 代码:

```
var settings = { validate: false, limit: 5, name: "foo" };  
var options = { validate: true, name: "bar" };  
jQuery.extend(settings, options);
```

结果:

```
settings = { validate: true, limit: 5, name: "bar" }
```

描述:

合并 defaults 和 options, 不修改 defaults。

jQuery 代码:

```
var empty = {};  
var defaults = { validate: false, limit: 5, name: "foo" };  
var options = { validate: true, name: "bar" };  
var settings = jQuery.extend(empty, defaults, options);
```

结果:

```
settings = { validate: true, limit: 5, name: "bar" }  
empty == { validate: true, limit: 5, name: "bar" }
```

Layui 数据表格的 update

更新指定行，重新渲染该行（可更新数据）。

不会重新请求，但是会刷新

```
data.hasRead = 'true';  
// 得到用户邮件状态（未读、已读）数量  
$.get("postbox/getMessageState", function(data) {  
    // 会将未读邮件、已读邮件的数量放进全局缓冲中  
    moduleCache.data.unReadMessageNum = data.unReadMessageNum ? data.unReadMessageNum : 0;  
    moduleCache.data.readMessageNum = data.readMessageNum ? data.readMessageNum : 0;  
    // 在邮件模块全局缓存数据中取出未读邮件数量  
    var unReadMessageNum = moduleCache.data.unReadMessageNum;  
    // 邮件模块index页面左侧功能栏的[未读邮件]数量显示修改  
    if (unReadMessageNum > 0) {  
        moduleCache.data.unReadMessageNum = unReadMessageNum;  
        inboxFalse.text("未读邮件(" + moduleCache.data.unReadMessageNum + ")");  
    } else {  
        inboxFalse.text("未读邮件");  
    }  
});  
obj.update && obj.update(data);
```

百度富文本编辑器 Ueditor

ue.setContent(content, true) , true 表示在原有的基础上追加内容

layui 数据表格的 event

监听工具条(table.on("tool(tableId)", function() {}))

```

    }, {title: "回复", align: "center", event: "useGoReply", templet: function() {
        var html = "";
        html += "<input type='checkbox' lay-skin='switch' lay-text='ON|OFF' checked />";
        return html;
    }}
    , {title: "转发", align: "center", event: "useGoForward", templet: function() {
        var html = "";
        html += "<input type='checkbox' lay-skin='switch' lay-text='ON|OFF' checked />";
        return html;
    }}
    , {title: "操作", align: "center", templet: function() {
        var html = "";
        html += '<div><a class="layui-btn layui-btn-xs" lay-event="signatureEdit">编辑</a>';
        html += '<a class="layui-btn layui-btn-danger layui-btn-xs" lay-event="signatureDelete">删除</a>';
        return html;
    }}
  ]
});

```

// 监听个性签名数据表格的事件

```

table.on("tool(signatureList)", function(obj) {
    var data = obj.data;

    switch (obj.event) {
        case "useGoNewWrite":
            console.log("--- useGoNewWrite ---");
            break;
        case "useGoReply":
            console.log("--- useGoReply ---");
            break;
        case "useGoForward":
            console.log("--- useGoForward ---");

```

//监听工具条

```
table.on('tool(test)', function(obj){ //注: tool是工具条事件名, test是table原始容器的属性 lay-filter="对应的值"  
    var data = obj.data; //获得当前行数据  
    var layEvent = obj.event; //获得 lay-event 对应的值 ( 也可以是表头的 event 参数对应的值 )  
    var tr = obj.tr; //获得当前行 tr 的DOM对象  
  
    if(layEvent === 'detail'){ //查看  
        //do something  
    } else if(layEvent === 'del'){ //删除  
        layer.confirm('真的删除行么', function(index){  
            obj.del(); //删除对应行 ( tr ) 的DOM结构, 并更新缓存  
            layer.close(index);  
            //向服务端发送删除指令  
        });  
    } else if(layEvent === 'edit'){ //编辑  
        //do something  
  
        //同步更新缓存对应的值  
        obj.update({  
            username: '123'  
            ,title: 'xxx'  
        });  
    }  
});
```


jQuery 中 empty 和 remove 的区别之一是

移除元素时包不包括自己

jQuery 中 attr 和 prop 的区别

固有属性：标签元素本身的属性，

自定义属性：自己定义的属性，如 age=" 21" ，age 就是自定义属性

- 对于 HTML 元素本身就带有的固有属性，在处理时，使用 prop 方法。
- 对于 HTML 元素我们自己自定义的 DOM 属性，在处理时，使用 attr 方法。

再举一个例子：

```
<input id="chk1" type="checkbox" />是否可见  
<input id="chk2" type="checkbox" checked="checked" />是否可见
```

像checkbox，radio和select这样的元素，选中属性对应“checked”和“selected”，这些也属于固有属性，因此需要使用prop方法去操作才能获得正确的结果。

```
$("#chk1").prop("checked") == false  
$("#chk2").prop("checked") == true
```

如果上面使用attr方法，则会出现：

```
$("#chk1").attr("checked") == undefined  
$("#chk2").attr("checked") == "checked"
```

@Lob 注解

[@Lob]

指定持久属性或字段应作为大对象持久保存到数据库支持的大对象类型。

映射到数据库Lob类型时，便携式应用程序应使用Lob注释。

当元素集合值是基本类型时，Lob注释可以与Basic注释或ElementCollection注释一起使用。

Lob可以是二进制或字符类型。


Lob类型是根据持久性字段或属性的类型推断的，除了字符串和基于字符的类型以外，默认为Blob。

String 类的默认值 为 longtext

byte[] Image File 默认值 为 longblob

lob (存储大对象的数据类型)

 编辑

 本词条缺少**名片图**，补充相关内容使词条更完整，还能快速升级，赶紧来**编辑**吧！

LOB(large object)是一种用于存储大对象的数据类型

| | | | |
|-----|--------------|------|------------------|
| 外文名 | lob | 其他缩写 | line-of-business |
| 类 别 | 用于存储大对象的数据类型 | 全 称 | LineOf Balance |

LOB(large object)是一种用于存储大对象的数据类型，如医学记录（如X-射线）、视频、图像等。LOB有三种类型：

BLOB：Binary Large Object、**CLOB**：Character Large Object、**DBCLOB**：Double-byte Character Large Object。每个LOB可以有2GB。

jQuery 中判断是否含有某个类

1. is(".classname")
2. hasClass("classname")

使用 layui 时要注意哪些问题，**比如**使用 form 表单那块，那么就记得渲染.

```
layui.use( "form" , function() {  
    var form = layui.form;  
    Form.render();  
});
```

[@Param](#)

@Param注解的作用是给参数命名,参数命名后就能根据名字得到参数值,正确的将参数传入sql语句中

富文本编辑器会对输入的内容进行格式处理。这是一个**细节**，假如你现在在做着长字符串替换的事情。

在富文本编辑器(Ueditor)会对输入的内容进行格式处理，需要进行文字匹配时要以编辑器处理过的格式进行匹配。

jQuery 的 on 用于动态绑定事件

```
$("#primary").on("click", "p", function() {})
```

那么这里给 p 绑定了点击事件

```
<div id="primary"><p></p></div>
```

jQuery 操作 select (取值、回显数据)

```
<select id=" a"  onchange=" change()" ><option value=" 1" >1</option> </select>
```

获取选中的 option 值

```
$( "#a" ).val();
```

<https://www.cnblogs.com/zhangym118/p/5509673.html>

点击 option 时触发 , 绑定 onchange 事件。

```
function change() {}
```

或者 `$("#a option:selected").val();`

bootstrap 表格重载

```
$("#id").bootstrapTable('refresh',{url:'url'});
```

jQuery trigger()

点击当前元素可触发另一元素某事件

触发 <input> 元素的 select 事件：

```
$("#button").click(function(){  
    $("#input").trigger("select");  
});
```

尝试一下 »

定义和用法

trigger() 方法触发被选元素上指定的事件以及事件的默认行为（比如表单提交）。

该方法与 [triggerHandler\(\)](#) 方法类似，不同的是 triggerHandler() 不触发事件的默认行为。

与 triggerHandler() 方法相比的不同之处：

- 它不会引起事件（比如表单提交）的默认行为
- .trigger() 会操作 jQuery 对象匹配的所有元素，而 .triggerHandler() 只影响第一个匹配元素。
- 由 .triggerHandler() 创建的事件不会在 DOM 树中冒泡；如果目标元素不直接处理它们，则不会发生任何事情。

Mapper 中使用模糊查询

LIKE 关键字配合使用

1、直接在传入 xml 前进行拼接(“_” 或者 “%”)

LIKE #{field}

2、xml 中使用 CONCAT 进行拼接

LIKE CONCAT("%", #{field}, "%")

3、使用单引号进行拼接

LIKE '%field%'

onerror 事件

jquery 的 error 事件。

如果 image 元素遇到错误，把它替换为文本：

```
$("#img").error(function(){  
    $("#img").replaceWith("<p>图片加载错误!</p>");  
});
```

尝试一下 »

定义和用法



使用on或者one来
绑定

`error()` 方法在 jQuery 版本 1.8 中被废弃。

当元素遇到错误时（当元素没有正确载入时），发生 `error` 事件。

`error()` 方法触发 `error` 事件，或规定当发生 `error` 事件时运行的函数。

提示：该方法是 `bind('error', handler)` 的简写方式。

Thymeleaf 模板引擎

`th:selected="${used.defaultmailbox != null} and ${used.defaultmailbox} eq ${semail.id}"`

- **简单表达式 (simple expressions)**

`${...}` 变量表达式

`*{...}` 选择变量表达式

`#{...}` 消息表达式

`@{...}` 链接url表达式

- **字面量**

`'one text','another one!','...'` 文本

`0,34,3.0,12.3,...` 数值

`true false` 布尔类型

`null` 空

`one,sometext,main` 文本字符

- **文本操作**

`+` 字符串连接

`[The name is ${name}]` 字符串连接

- **算术运算**

`+, -, *, / , %` 二元运算符

`-` 负号 (一元运算符)

- **布尔操作**

`and,or` 二元操作符

`!,not` 非 (一元操作符)

- **关系操作符**

`> , < , >= , <=` (gt , lt , ge , le)

`= , !=` (eq, ne)

- **条件判断**

(if) ? (then) if-then

(if) ? (then) : (else) if-then-else

基本语法

<https://www.cnblogs.com/nuoyiamy/p/5591559.html>

@ResponseBody

@ResponseBody 注解的作用是将 controller 的方法返回的对象通过适当的转换器转换为指定的格式之后，写入到 response 对象的 body 区，通常用来返回 JSON 数据或者是 XML 数据，需要注意的是，在使用此注解之后不会再走视图处理器，而是直接将数据写入到输入流中，他的效果等同于通过 response 对象输出指定格式的数据。

效果等同于如下代码：

```
@RequestMapping("/login")
public void login(User user, HttpServletResponse response){
    response.getWriter().write(JSONObject.fromObject(user).toString());
}
```

[layui 表单重置, js 重置表单, jquery 重置表单](#)

```
$("#bindServiceEmailForm")[0].reset();
```

Layui form 数据回显

// 数据邮箱信息回显

```
form.val("bindServiceEmail", {  
    "emailAccount": data.account  
    , "emailPassword": data.password  
    , "protocol": data.protocol  
    , "host": data.host  
    , "port": data.port          "name": value  
    , "isSSL": false  
    , "sendHost": data.send_host  
    , "sendPort": data.send_port  
})
```

[jquery offset\(\)方法](#)

定义和用法

offset() 方法设置或返回被选元素相对于文档的偏移坐标。

当用于返回偏移时：

该方法返回第一个匹配元素的偏移坐标。它返回一个带有两个属性（以像素为单位的 top 和 left 位置）的对象。

当用于设置偏移时：

该方法设置所有匹配元素的偏移坐标。



ibatis 返回整形时，若无数据则返回 0

jQuery 遍历对象

[对元素进行遍历，](#)

```
$("#元素").each(callback);
```

```
如：$("#div").each(function(index) {  
  })
```

[·对数组进行遍历](#)

```
$( "数组" ).each(callback);
```

```
如：$(data, function(index, item) {  
  })
```

[thymeleaf 的 each 用法](#)

除了遍历，我们还可以拿到遍历的其他信息，比如下标，单行双行，这时要有一状态变量，假设命名为：iterStat，那么从iterStat中可获取的信息如下：

// 来自官网：

当前迭代索引，从0开始。这是index属性。

当前迭代索引，从1开始。这是count属性。

迭代变量中的元素总数。这是size属性。

每个迭代的iter变量。这是current属性。

目前的迭代是偶数还是奇数。这些是even/odd布尔属性。

目前的迭代是否是第一个。这是first布尔属性。

目前的迭代是否是最后一个。这是last布尔属性。

看html:

```
1 <table border="1">
2   <tr th:each="m,iterStat : ${list}">
3       <td th:text="下标 : ${iterStat.index} 下标是不是单数 : ${iterStat.odd}"/>
4       <td th:text="${m.name}"/>
5       <td th:text="${m.age}"/>
6   </tr>
7 </table>
```

使用 JQuery 查看祖先元素

parents()

描述:

找到每个span元素的所有祖先元素。

HTML 代码:

```
<html><body><div><p><span>Hello</span></p><span>Hello Again</span></div></body></html>
```

<http://jquery.cuishifeng.cn/parents.html>

parentsUntil

查找当前元素的所有父辈元素，直到遇到匹配的那个元素为止。

如果提供的 jQuery 代表了一组 DOM 元素，`.parentsUntil()`方法也能让我们找遍所有元素的祖先元素，**直到**遇到了一个跟提供的参数匹配的元素的时候**才会停下来**。这个返回的 jQuery 对象里**包含了下面所有找到的父辈元素，但不包括那个选择器匹配到的元素**。

描述:

查找item-a的祖先，但不包括level-1

HTML 代码:

```
<ul class="level-1">
  <li class="item-i">I</li>
  <li class="item-ii">II
    <ul class="level-2">
      <li class="item-a">A</li>
      <li class="item-b">B
        <ul class="level-3">
          <li class="item-1">1</li>
          <li class="item-2">2</li>
          <li class="item-3">3</li>
        </ul>
      </li>
      <li class="item-c">C</li>
    </ul>
  </li>
  <li class="item-iii">III</li>
</ul>
```

jQuery 代码:

```
$('.li.item-a').parentsUntil('.level-1')
  .css('background-color', 'red');
```

<http://jquery.cuishifeng.cn/parentsUntil.html>

thymeleaf 的 th:text="|字符拼接的方式输出\${user.username}|"

或者 th:text=""字符拼接方式输出'+ \${user.username}"

th:unless , 和 th:if 是相反的。如果为 false 则执行

2.3. 条件语法 : th:if; th:unless

演示如下功能

- th:if : 如果值是true , 则打印整个节点
- th:unless: 和th:if是相反功能 , 如果值为false,则打印整个节点

```
1      <!-- th:if : 如果值是true , 则打印<span>整个节点 -->
2      <span th:if="${user.isAdmin}" th:text="${user.name} + '是管理员'">   </span><br />
3      <!-- th:unless: 和th:if是相反功能 , 如果值为false,则打印<span>整个节点 -->
4      <span th:unless="not ${user.isAdmin}" th:text="${user.name} + '是管理员'">   </span><br />
```

<https://blog.csdn.net/hry2015/article/details/73253080>

Swagger

是一个开源项目，一个 API 文档生成工具，可用于生成 API 文档、维护和调试接口。

Swagger是一组开源项目，其中主要项目如下：

1. **Swagger-tools**:提供各种与Swagger进行集成和交互的工具。例如模式检验、Swagger 1.2文档转换成Swagger 2.0文档等功能。
2. **Swagger-core**: 用于Java/Scala的的Swagger实现。与JAX-RS(Jersey、Resteasy、CXF...)、Servlets和Play框架进行集成。
3. **Swagger-js**: 用于JavaScript的Swagger实现。
4. **Swagger-node-express**: Swagger模块，用于node.js的Express web应用框架。
5. **Swagger-ui**：一个无依赖的HTML、JS和CSS集合，可以为Swagger兼容API动态生成优雅文档。
6. **Swagger-codegen**：一个模板驱动引擎，通过分析用户Swagger资源声明以各种语言生成客户端代码。

<https://blog.csdn.net/i6448038/article/details/77622977>

bootstrap 表格带参数刷新

```

function tableRefresh(param) {
  let condition = {
    query: {
      "userId": param.userId,
      "checkSource": param.checkSource,
      "checkTime": param.checkTime
    }
  };
  $("#checkCustomerTable").bootstrapTable('refresh', condition);
}

```

参数

table的id

refresh

- **Parameter:** `params`

- **Detail:**

Refresh the remote server data, you can set `{silent: true}` to **refresh** the data silently, and set `{url: newUrl, pageNumber: pageNumber, pageSize: pageSize}` to change the url (optional), page number (optional) and page size (optional). To supply query params specific to this request, set `{query: {foo: 'bar'}}`.

<https://blog.csdn.net/lanyang123456/article/details/55805478>

MyBatis 的“ if-else” 写法

```
<insert id="insertBusinessUserList" parameterType="java.util.List">
  insert into `business_user` (`id` , `user_type` , `user_login` )
  values
  <foreach collection="list" index="index" item="item" separator=",">
    <trim prefix="(" suffix=")" suffixOverrides=",">
      <choose>
        <when test="item.id != null and item.id != ''">
          #{item.id,jdbcType=CHAR},
        </when>
        <otherwise>
          '',
        </otherwise>
      </choose>
      <choose>
        <when test="item.userType != null and item.userType != ''">
          #{item.userType,jdbcType=VARCHAR},
        </when>
        <otherwise>
          '',
        </otherwise>
      </choose>
    </trim>
  </foreach>
</insert>
```

<https://www.cnblogs.com/a8457013/p/8033263.html>

bootstraps 带参数刷新

data-query-params="tablequeryparams"

function tablequeryparams(params) {

let queryParams = {

limit: params.limit, //页面大小

offset: params.offset, //页码

sort: params.sort, //排序列名

order: params.order, //排位命令 (desc , asc)

userId: \$("#userId").val(),

checkSource: \$("#checkSource").val(),

checkTime: \$("#checkTime").val()

```
};  
return queryParams;  
}
```

jquery 的 ready()与 window.onload()

尽管两者都是等到页面加载完成后再执行，但是，前者只需要加载完 DOM 结构即可执行，后者则是加载时更彻底些(包括图片等)。

| | window.onload() | \$(document).ready() |
|------|---|---|
| 加载时机 | 必须等待网页全部加载完毕（包括图片等），然后再执行JS代码 | 只需要等待网页中的DOM结构加载完毕，就能执行JS代码 |
| 执行次数 | 只能执行一次，如果第二次，那么第一次的执行会被覆盖 | 可以执行多次，第N次都不会被上一次覆盖 |
| 举例 | <p>以下代码无法正确执行：</p> <pre> window.onload = function() { alert("text1");}; window.onload = function() { alert("text2");}; </pre> <p>结果只输出第二个</p> | <p>以下代码正确执行：</p> <pre> \$(document).ready(function() {alert("Hello")}); \$(document).ready(function() {alert("Hello")}); </pre> <p>结果两次都输出</p> |
| 简写方案 | 无 | \$(function () {}) |

jquery 的 ready() 与 window.onload()的区别

获取上一个父节点 [parent\(\[expr\]\)](#)

获取兄弟节点 [prev\(\[expr\]\)](#)

(expr:增加表达式进行筛选)

移除全部样式

可以使用 [removeAttr\("style" \)](#) , 这样

removeAttr(name)

概述

从每一个匹配的元素中删除一个属性

1.6以下版本在IE6使用jQuery的removeAttr方法删除disabled是无效的。解决的方法就是使用\$("XX").prop("disabled",false);
1.7版本在IE6下已支持删除disabled。

参数

| name | String |
|---------|--------|
| 要删除的属性名 | |

移除单个样式

可以使用

css("样式名" , "");

jQuery Validation Engine 插件，用于验证表单

<from id=" fromId" >

```
<input type=" text" class=" validate[required,maxSize[16]]"
</from>
```

如果验证失效，则在进行操作时再校验一次。调用

`$("#fromId").validationEngine("validate")` , 符合条件 返回 true

jQuery 绑定多个事件，执行同一方法或者执行不同方法

```
$( "div" ).on( "click keypress" , function() {});
```

```
$( "div" ).on({
    click: function() {},
    keypress: function() {}
})
```

<https://blog.csdn.net/pangchengyong0724/article/details/53186127>

jQuery(鼠标按下、鼠标松开、鼠标左键点击、鼠标双击)事件

mousedown、mouseup、click、dbclick

<https://www.jb51.net/article/71712.htm>

Spring-SpringMVC-MyBatis 整合

- 1.web.xml (SpringIoC 初始化、配置文件加载、servlet 注册、请求映射)
- 2.spring-mvc.xml (包扫描、注解驱动、视图解析器)
- 3.spring-mybatis.xml (包扫描、数据源配置 (先加载属性文件) 、SqlSessionFactory 配置、dao 接口映射)
4. 因为 spring-mybatis.xml 中的 SqlSessionFactory 配置中用到了 mybatis 的配置文件，所以要写下 mybatis-config.xml。

注：

使用 classpath 书写文件路径时禁止带空格，否则报错

Caused by: org.xml.sax.SAXParseException; lineNumber: 1; columnNumber: 1; 前言中不允许有内容。

web.xml

listener

web容器初始化时初始ioC容器（ContextLoaderListener）

context-param

加载一些配置文件，如spring与mybatis整合的文件，应用范围是application

servlet

配置标准的servlet（DispatcherServlet），它会自己初始化一个ioC容器，但是会先获取根ioC，把根的作为parent再初始化自己的。一般根ioC是整个web应用的组件，如Service、dao的对象，自己的一般跟该servlet相关的，如controller、viewresovler

servlet-mapping

url映射

spring-mvc.xml

context:component-scan

注解扫描，如controller中的注解

mvc:annotation-driven

注解驱动，如方法参数自动绑定、验证

bean (InternalResourceViewResolver)

视图解析器。
需要注入两个参数：视图名称前缀、视图名称后缀

spring-mybatis.xml

context:component-scan

注解扫描，如service中的注解

bean (PropertyPlaceholderConfigurer)

数据源连接之读取jdbc属性文件 (properties)。
注入参数：locations

bean (DriverManagerDataSource)

数据源连接之连接。
注入四个参数：
driverClassName、url、
username、password

bean (SqlSessionFactoryBean)

sqlSessionFactory的创建，需要注入三个参数：
configLocation、mapperLocations、dataSource

bean (MapperScannerConfigurer)

dao接口映射。用于创建Dao实现类，注入两个属性：
basePackage、sqlSessionFactoryBeanName

【Spring】浅谈 ContextLoaderListener 及其上下文与 DispatcherServlet 的区别

创建 springboot 的三种方式。

1.在线

<https://start.spring.io>

2.通过 IDE

IDEA 或者 STS (SpringToolSuit , Spring 整的开发工具,在 eclipse 的基础上做了些优化,让开发 spring 项目时更爽) 都可以。

3.通过 maven 项目

新建 maven 项目 , 添加相关 springboot 依赖 , 创建启动类

[创建一个 Spring Boot 项目 , 你会几种方法 ?](#)

springboot 屏蔽内置 tomcat

- 1、修改打包方式
- 2、屏蔽内置 tomcat , 两种方式
- 3、修改启动类 , 继承 SpringBootServletInitializer , 重写 configure(SpringApplicationBuilder)方法
- 4、使用 eclipse 进行打包 (war) , 右键项目 , maven clean , maven install

1、<packaging>war</packaging>

2、

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <!-- 去除springboot内置tomcat -->
  <exclusions>
    <exclusion>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-tomcat</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

或者这样

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-tomcat</artifactId>
  <scope>provided</scope>
</dependency>
```

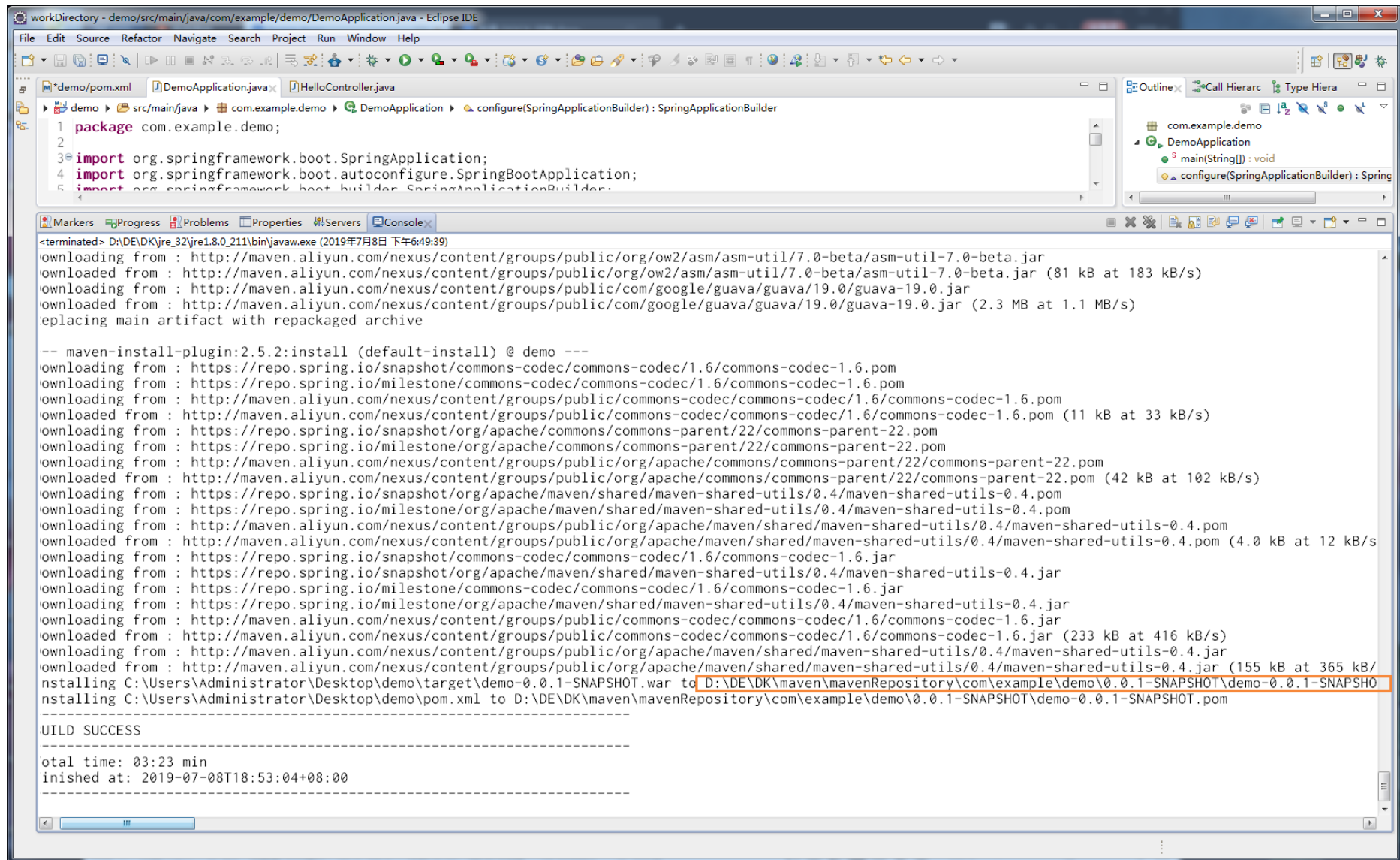
3、

```
@SpringBootApplication
public class DemoApplication extends SpringBootServletInitializer {

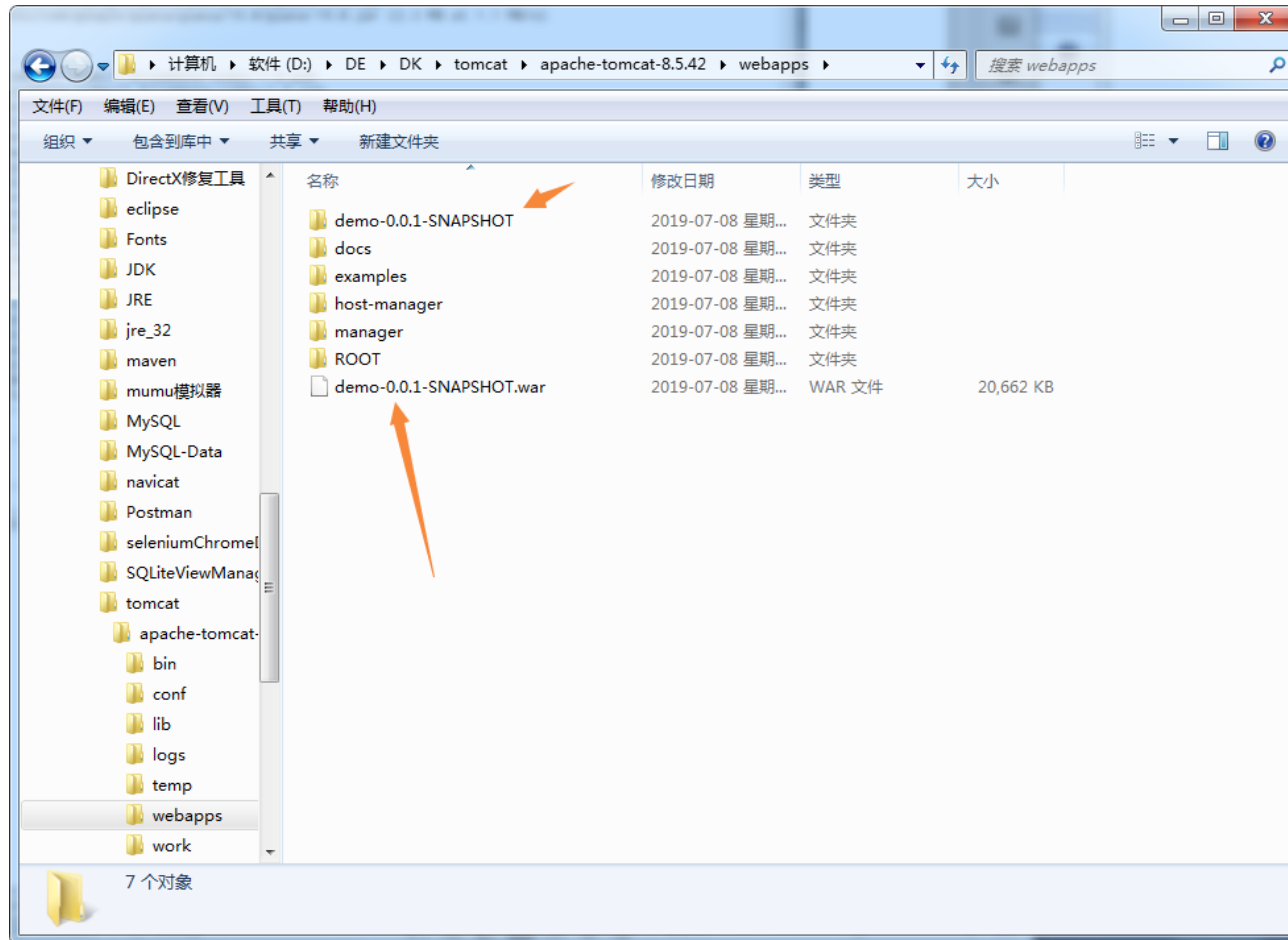
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {
        return builder.sources(this.getClass());
    }
}
```

4、



5、放到 tomcat 的 webapps 目录下，tomcat 启动时自动解压



6、访问 <http://localhost:8080/demo-0.0.1-SNAPSHOT> 或其目录下路径

