**编码流程**

1. 编辑页面通用行为js对象

**属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **属性名** | | **类型** | **含义** |
| Data | | Array | 对应一条待编辑记录中的各项。 |
| Data[i] | Key |  |  |
| Key\_Change |  | 页面对应展示该数据的标签id |
| Value |  |  |
| Type |  |  |
| Mode |  |  |
| Name |  |  |
| CheckType |  |  |
|  | |  |  |

**方法**

|  |  |  |
| --- | --- | --- |
| **方法签名** | **参数说明** | **返回值类型** |
| function Fun\_Data\_Init(obj) | obj | — |
| 方法逻辑说明 | if (obj == null || obj == "") {  return;  }  // 如果传入的是数据集，则取第一个数据  while (true) {  var len = obj.length;  if (len != null && len > 0) {  obj = obj[0];  break;  }  break;  }  var mydata = new Array();  $.each(obj, function (val1, val2) {  // Type 1 td/2 text等  // Mode 0 初始/1 强制赋值  mydata.push({ "Key": val1  , "Key\_Change": val1  , "Value": val2  , "Type": ""  , "Mode": "0" });  });  this.Data = mydata; | |
| function Fun\_Data\_Add | val1  val2 | — |
| 方法逻辑说明 | for (var i = 0; i < this.Data.length; i++) {  var obj = this.Data[i];  if (obj.Key == val1) {  obj.Value = val2;  obj.Mode = "1";  return false;  }  }  this.Data.push({ "Key": val1, "Key\_Change": val1, "Value": val2, "Type": "", "Mode": "1" }); | |
| function Fun\_Change\_Key (change\_l, change\_r) | change\_l  change\_r | — |
| 方法逻辑说明 | for (var i = 0; i < this.Data.length; i++) {  var obj = this.Data[i];  var id = obj.Key;  if (change\_l != null && change\_l != "") {  id = change\_l + id;  }  if (change\_r != null && change\_r != "") {  id = id + change\_r;  }  obj.Key\_Change = id;  } | |
| function Fun\_Set\_Html |  |  |
| 方法逻辑说明 | for (var i = 0; i < this.Data.length; i++) {  var obj = this.Data[i];  var id = obj.Key\_Change;  if ($("#" + id).is("td")) {  $("#" + id).html(obj.Value);  obj.Type = "1";  } else {  var \_type = $("#" + id).prop("type");  if (\_type != null) {  \_type = \_type.toLowerCase();  if (\_type == "text" || \_type == "hidden" || \_type == "password" || \_type == "select-one") {  $("#" + id).val(obj.Value);  obj.Type = "2";  }  }  }  }   1. 将数据展示在页面上，根据Data数组，并设置数组元素的Type属性。 | |
| Fun\_Get\_Html |  |  |
| 方法逻辑说明 | for (var i = 0; i < this.Data.length; i++) {  var obj = this.Data[i];  var id = obj.Key\_Change;  if ($("#" + id).is("td")) {  obj.Value = $("#" + id).html();  obj.Type = "1";  } else {  var \_type = $("#" + id).prop("type");  if (\_type != null) {  \_type = \_type.toLowerCase();  if (\_type == "text" || \_type == "hidden" || \_type == "password" || \_type == "select-one") {  obj.Value = $("#" + id).val();  obj.Type = "2";  }  }  }  }  1、获取页面值，存入Data数组，并设置元素Type值 | |
| Fun\_Serialize\_All |  | String  返回查询字符串 |
| 方法逻辑说明 | var ret = "";  for (var i = 0; i < this.Data.length; i++) {  var obj = this.Data[i];  if (obj.Type == "1") {  if (ret != "") {  ret += "&";  }  ret += obj.Key + "=" + $("#" + obj.Key\_Change).html();  } else if (obj.Type == "2") {  if (ret != "") {  ret += "&";  }  ret += obj.Key + "=" + $("#" + obj.Key\_Change).val();  } else if (obj.Mode == "1") {  if (ret != "") {  ret += "&";  }  ret += obj.Key + "=" + obj.Value;  } else {  if (ret != "") {  ret += "&";、  }  ret += obj.Key + "=" + obj.Value;  }  }  return ret;  1、将Data数组全部元素，拼接成查询字符串 | |
| Fun\_Serialize |  | String  返回查询字符串 |
| 方法逻辑说明 | var ret = "";  for (var i = 0; i < this.Data.length; i++) {  var obj = this.Data[i];  if (obj.Type == "1") {  if (ret != "") {  ret += "&";  }  ret += obj.Key + "=" + $("#" + obj.Key\_Change).html();  } else if (obj.Type == "2") {  if (ret != "") {  ret += "&";  }  ret += obj.Key + "=" + $("#" + obj.Key\_Change).val();  } else if (obj.Mode == "1") {  if (ret != "") {  ret += "&";  }  ret += obj.Key + "=" + obj.Value;  }  }  return ret;  1、将Data数组部分元素，拼接成查询字符串 | |
| Fun\_Serialize\_Para | ids |  |
| 方法逻辑说明 | var list = ids.split(',');  var ret = "";  for (var i = 0; i < this.Data.length; i++) {  for (var j = 0 ; j < list.length; j++) {  if (ret != "") {  ret += "&";  }  if (this.Data[i].Key == list[j]) {  ret += this.Data[i].Key + "=" + this.Data[i].Value;  }  }  }  return ret;  1、将Data数组指定Key，拼接成查询字符串 | |
| Fun\_Get\_OneValue | key |  |
| 方法逻辑说明 | for (var i = 0 ; i < this.Data.length; i++) {  if (this.Data[i].Key == key) {  return this.Data[i].Value;  }  }  return "";  1、取值 | |
| Fun\_Change\_OneValue | key, val |  |
| 方法逻辑说明 | for (var i = 0 ; i < this.Data.length; i++) {  if (this.Data[i].Key == key) {  this.Data[i].Value = val;  }  }  return "";  1、设置值 | |
| Fun\_Set\_OneCheckInfo | key,  name,  checkType |  |
| 方法逻辑说明 | var tf = false;  if (this.Data == null || this.Data.length == 0) {  this.Data = new Array();  }  for (var i = 0 ; i < this.Data.length; i++) {  if (this.Data[i].Key == key) {  this.Data[i].Name = name;  this.Data[i].CheckType = checkType;  tf = true;  return;  }  }  if (!tf) {  this.Data.push({ "Key": key, "Key\_Change": key, "Value": "", "Type": "", "Mode": "0", "Name": name, "CheckType": checkType });  return;  }  return; | |
| Fun\_Check |  |  |
|  | for (var i = 0; i < this.Data.length; i++) {  var obj = this.Data[i];  if (obj.Name == null || obj.Name == "") {  continue;  }  var checkType = obj.CheckType.split(',');  for (var j = 0 ; j < checkType.length; j++) {  var val = checkType[j];  val = CommonFun\_ReplaceAll(val, " ", "").toLowerCase();  if (val == "null") {  if (obj.Value == null || obj.Value == "") {  throw obj.Name + "不能为空！";  }  }  }  } | |
|  |  |  |
|  |  |  |
|  |  |  |

1. 分页列表页面通用行为js对象(PagingJsObject)

**属性**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **属性名** | | | **类型** | **含义** |
| Guid | | | String | 对象标识 |
| PageList\_Index | | | Int | 当前页页码 |
| PageList\_Count | | | Int | 每页记录数 |
| UserID | | | String | 用户ID |
| OrgCode | | | String | 用户所属组织机构ID |
| RRoleCode | | | String | 用户角色Code |
| DRoleType | | | String | 用户角色类型 |
| Para1 | | | String | 备用参数1 |
| Para2 | | | String | 备用参数2 |
| Para3 | | | String | 备用参数3 |
| Para4 | | | String | 备用参数4 |
| Para5 | | | String | 备用参数5 |
| Para6 | | | String | 备用参数6 |
| Para7 | | | String | 备用参数7 |
| Para8 | | | String | 备用参数8 |
| Para9 | | | String | 备用参数9 |
| PageList\_Data | | | Array | 当前页面数据集 |
| PageList\_RowCount | | | Int | 列表的总条数 |
| PageList\_Num | | | Array | 在页面上展示的页码数组 |
| PageList\_BtnInfo | | | Object | 列表操作列对象 |
| PageList\_BtnInfo | Data | | Array | 当前分页页面数据集数组  与PageList\_Data一致 |
| BtnI | | Int | 操作列按钮数 |
| Btns | | Array | 操作列按钮定义集合 |
| PageList\_BtnInfo | Btns | Tag | String | 按钮备注 |
| Title | String | 按钮标题 |
| Fun\_Callback | Function | 单击事件回调函数 |
| Fun\_Click | Function | 单击事件处理函数 |
|  | | |  |  |

**方法**

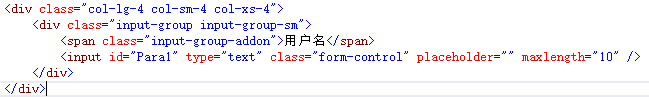
|  |  |  |
| --- | --- | --- |
| **方法签名** | **参数说明** | **返回值类型** |
| function PageList\_Json | — | String |
| 方法逻辑说明 | var json = "";  json += "PageList\_Index=" + this.PageList\_Index;  json += "&PageList\_Count=" + this.PageList\_Count;  json += "&UserID=" + this.UserID;  json += "&OrgCode=" + this.OrgCode;  json += "&RRoleCode=" + this.RRoleCode;  json += "&DRoleType=" + this.DRoleType;  if ($("#Para1").val() != null) {  this.Para1 = $("#Para1").val();  }  if ($("#Para2").val() != null) {  this.Para2 = $("#Para2").val();  }  if ($("#Para3").val() != null) {  this.Para3 = $("#Para3").val();  }  if ($("#Para4").val() != null) {  this.Para4 = $("#Para4").val();  }  if ($("#Para5").val() != null) {  this.Para5 = $("#Para5").val();  }  if ($("#Para6").val() != null) {  this.Para6 = $("#Para6").val();  }  if ($("#Para7").val() != null) {  this.Para7 = $("#Para7").val();  }  if ($("#Para8").val() != null) {  this.Para8 = $("#Para8").val();  }  if ($("#Para9").val() != null) {  this.Para9 = $("#Para9").val();  }  if (this.Para1 != "") {  json += "&Para1=" + this.Para1;  }  if (this.Para2 != "") {  json += "&Para2=" + this.Para2;  }  if (this.Para3 != "") {  json += "&Para3=" + this.Para3;  }  if (this.Para4 != "") {  json += "&Para4=" + this.Para4;  }  if (this.Para5 != "") {  json += "&Para5=" + this.Para5;  }  if (this.Para6 != "") {  json += "&Para6=" + this.Para6;  }  if (this.Para7 != "") {  json += "&Para7=" + this.Para7;  }  if (this.Para8 != "") {  json += "&Para8=" + this.Para8;  }  if (this.Para9 != "") {  json += "&Para9=" + this.Para9;  }  return json;  将对象属性名与属性值拼接成查询字符串返回。 | |
| function Fun\_Loading | — | — |
| 方法逻辑说明 | 加载当前分页数据集 | |
| function Fun\_InitData(data) | data：Array  当前页面数据集  每个元素都是一个对象，且对象中包含属性c，代表记录总条数 | — |
| 方法逻辑说明 | if (data == null || data.length == 0) {  if (this.PageList\_Index > 1) {  this.PageList\_Index--;  this.Fun\_Loading();  return false;  } else {  this.PageList\_Data = null;  this.PageList\_RowCount = 1;  }  } else {  this.PageList\_Data = data;  if (data[0].c != null && parseFloat(data[0].c) > 0) {  this.PageList\_RowCount = parseFloat(data[0].c);  } else {  this.PageList\_RowCount = 1;  }  }  this.Fun\_ResetList();  功能说明：   1. 根据参数data，重新设置PageList\_RowCount | |
| function Fun\_ResetList | — | — |
| 方法逻辑说明 | if (this.PageList\_RowCount <= 0) {  this.PageList\_Num = [];  } else {  var max = Math.ceil(1.00 \* this.PageList\_RowCount / this.PageList\_Count);  if (this.PageList\_Index > max) {  this.PageList\_Index = max;  return false;  }  var list = [];  for (var i = this.PageList\_Index - 5; i < this.PageList\_Index; i++) {  if (i > 0) {  list.push(i);  }  }  for (var i = this.PageList\_Index; i < this.PageList\_Index + 6 && i <= max; i++) {  list.push(i);  }  this.PageList\_Num = list;  }  功能说明：   1. 首先根据记录总条数，计算页码数，再根据当前页码数，计算PageList\_Num | |
| function Fun\_InitDiv (divID) | divID：String  页码html容器id |  |
| 方法逻辑说明 | 功能说明：   1. 根据PageList\_Num，生成展示页码html，并输出到参数divID指定的页码容器。 | |
| function Fun\_Bind\_Div | — | — |
| 方法逻辑说明 | 功能说明：   1. 绑定页码单击事件处理程序 |  |
| function Fun\_GoTo(ind) | Ind：int  页码 |  |
| 方法逻辑说明 | 功能说明：   1. 根据Ind，设置当前页码PageList\_Index，加载当前分页数据集。 | |
| function Fun\_InitTable(bodyID, key) | bodyID：string  表体容器id  key |  |
| 方法逻辑说明 | 功能说明：   1. 设置表体结构和数据 | |
| function Fun\_Bind\_Table | — | — |
| 方法逻辑说明 | 功能说明： | |
| function Fun\_GetObjValue(obj, id) |  |  |
| 方法逻辑说明 | 功能说明： | |
| function Fun\_GetOPHtml(ret, obj, btn) |  |  |
| 方法逻辑说明 | 功能说明： | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

1. 分页列表页面编程步骤

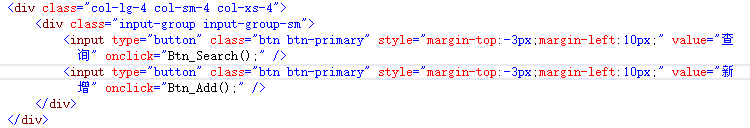
* 从控制器分页列表页面对应的方法，创建分页列表页面，分页列表页面基于通过模板【/Views/A02\_HomePage/\_MB.cshtml】
* 在新建的列表页面，设置页面标题ViewBag.Title
* 编写列表框架代码，如下图所示



以下对分页列表框架进行简单说明：



* 此处定义页面查询条件。
* 注意查询条件输入框id的命名：Para1、Para2……，这样命名为了是，与【分页列表页面通用行为js对象】中的Para1、Para2……相对应，PagingJsObject，提供PageList\_Json方法，调用该方法可将页面id为Para1、Para2……标签的值组合成查询字符串。类似Para1=[Para1]&Para2=[Para2]……



* 此处定义页面操作按钮【查询】【新增】



* 此处定义页面列表结构，页面列表分两块：列表展示区域、页码展示区域
* 列表展示区结构

<table>

<thead>

<tr>

<th name={name} class={l|c|r}>列名</th>

……

……

<th name=”op” ….>操作</th>

</tr>

</thead>

<tbody id=”tbodyid”></tbody>

</table>

* 列表列名定义

<th name={name} class={l|c|r}>列名</th>其中：

class={l|c|r}，分别单元格中内容的水平对齐方式。

name={name}，{name}必须与数据源中列名一致，因为分页列表页面通用行为js对象(PagingJsObject)，该对象有方法Fun\_InitTable，调用该方法，将使用数据源填充表格体（tbody）。分为：构造表体结构及填充数据两个步骤。

* 构造表体

function (bodyID) Fun\_InitTable

bodyID是页面表体容器tbodyid

根据页面定义的<th>，构造如下表体结构

<tr name= PagingJsObject .GUID >

<td {0}>{0}</td>

<td {0}>{0}</td>

……

//页面thead中定义了多少th，此处就有多少个td

<td {0}>{0}</td>

</tr>  
<tr name= PagingJsObject .GUID >

<td {1}>{1}</td>

<td {1}>{1}</td>

……

//页面thead中定义了多少th，此处就有多少个td

<td {1}>{1}</td>

</tr>

……

// PagingJsObject.PageList\_Count定义了一页多少条记录，此处就有多少个tr

<tr name= PagingJsObject .GUID >

<td {14}>{14}</td>

<td {14}>{14}</td>

……

//页面thead中定义了多少th，此处就有多少个td

<td {14}>{14}</td>

</tr>

* 填充数据

1. 填充数据放在PagingJsObject. PageList\_Data，此为一个对象数组，数组中每条元素对应一行记录。
2. 初始化PagingJsObject.PageList\_BtnInfo.Data为空数组
3. 循环每条填充数据，应用步骤（4）至（10）
4. 将PagingJsObject. PageList\_Data【i】. BtnI= i，为填充数据PagingJsObject. PageList\_Data数组中的每个元素对象增加一个BtnI属性，并设置为下标i。
5. 将填充数据PagingJsObject. PageList\_Data【i】，压入PagingJsObject.PageList\_BtnInfo.Data数组
6. 将PagingJsObject.PageList\_BtnInfo. BtnI设置为下标i。见补充说明
7. 循环每一列，应用步骤（8）至（10）
8. 获取每列定义的class属性，替换html中的{i}字符串，并根据列定义的name属性，查找数据集中对应的值。见补充说明
9. 如果该列是操作列，即name=“op”，取出PageList\_BtnInfo.Btns，这是一个对象数组，数组中的每个元素结构如下{Tag，Title，Fun\_CallBack,Fun\_Click}，循环这个对象数组，应用步骤（10）
10. 生成操作列按钮html。
11. 如果填充数据，少于行数PageList\_Count，处理非数据行。
12. 将处理好的html输出到页面tbody
13. 绑定列表操作列按钮单击事件。
14. 补充说明，表体html构建过程，中{i}的替换过程。

以第一行，第一列为例：

原html字符串如下：

<tr name= PagingJsObject .GUID >

<td {0}>{0}</td>

<td {0}>{0}</td>

……

//页面thead中定义了多少th，此处就有多少个td

<td {0}>{0}</td>

</tr>  
<tr name= PagingJsObject .GUID>

<td {1}>{1}</td>

<td {1}>{1}</td>

……

//页面thead中定义了多少th，此处就有多少个td

<td {1}>{1}</td>

</tr>

……

// PagingJsObject.PageList\_Count定义了一页多少条记录，此处就有多少个tr

<tr name= PagingJsObject .GUID>

<td {14}>{14}</td>

<td {14}>{14}</td>

……

//页面thead中定义了多少th，此处就有多少个td

<td {14}>{14}</td>

</tr>

**经过上述一次操作后html字符串编程**

<tr name= PagingJsObject .GUID>

**<td style='text-align:center;'>{0}</td>**

<td {0}>{0}</td>

……

//页面thead中定义了多少th，此处就有多少个td

<td {0}>{0}</td>

</tr>  
<tr name= PagingJsObject .GUID >

<td {1}>{1}</td>

<td {1}>{1}</td>

……

//页面thead中定义了多少th，此处就有多少个td

<td {1}>{1}</td>

</tr>

……

// PagingJsObject.PageList\_Count定义了一页多少条记录，此处就有多少个tr

<tr name= PagingJsObject .GUID >

<td {14}>{14}</td>

<td {14}>{14}</td>

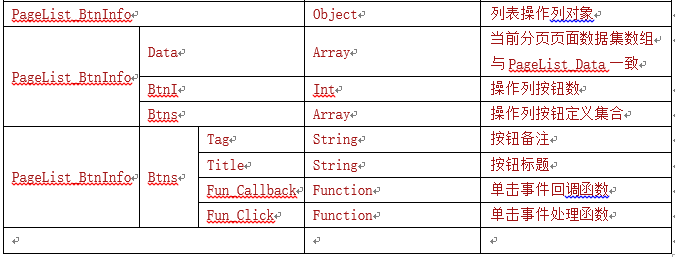
……

//页面thead中定义了多少th，此处就有多少个td

<td {14}>{14}</td>

</tr>

1. 补充说明：PagingJsObject.PageList\_BtnInfo



这个对象在PagingJsObject对象创建时，赋值，例如：

//创建PagingJsObject对象

var dataObj = PageListFun\_Data();

//给UserID，赋值当前用户ID

dataObj.UserID = "@ViewBag.UserID";

//给OrgCode，赋值当前用户所属组织机构Code

dataObj.OrgCode = "@ViewBag.OrgCode";

//给RRoleCode，赋值当前用户角色Code

dataObj.RRoleCode = "@ViewBag.RRoleCode";

//给DRoleType，赋值当前用户角色类型

dataObj.DRoleType = "@ViewBag.DRoleType";

//重新定义对象加载方法

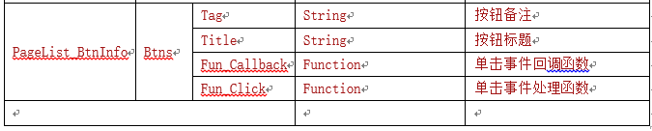
dataObj.Fun\_Loading = function () {

initHtml();

};

//给PageList\_BtnInfo.Btns数组初始化

//给PageList\_BtnInfo.Btns数组是一个对象数组，每个元素对象结构如下：



dataObj.PageList\_BtnInfo.Btns = [

{

"Tag": "edit",

"Title": "编辑",

"Fun\_Callback": function (obj) {

return true;

},

//定义编辑按钮单击事件处理程序

//参数obj，表示按钮所在行数据对象

//行数据对象中，包括ID字段

//点击【编辑】按钮，打开【编辑】页面，编辑页面关闭后，调用列表加载方法

"Fun\_Click": function (obj) {

var ID = obj.ID;

CommonFun\_OpenLayer({ "Type": "700\*550", "Title": "编辑", "Url": "/B10\_User/Edit", "Data": { "ID": ID }, "Fun\_End": function () { initHtml(); } });

}

}, {

"Tag": "update1",

"Title": "启用",

"Fun\_Callback": function (obj) {

if (obj.Del == "1") {

return true;

} else {

return false;

}

},

"Fun\_Click": function (obj) {

var ajaxObj = CommonFun\_Ajax();

ajaxObj.url = "../api/B10\_User/DoUpdate";

ajaxObj.data = { "para": "ID=" + obj.ID + "&Del=0" };

ajaxObj.getData(function (data) {

initHtml();

}, function () {

// Error

});

}

}, {

"Tag": "update2",

"Title": "作废",

"Fun\_Callback": function (obj) {

if (obj.Del == "0") {

return true;

} else {

return false;

}

},

"Fun\_Click": function (obj) {

var ajaxObj = CommonFun\_Ajax();

ajaxObj.url = "../api/B10\_User/DoUpdate";

ajaxObj.data = { "para": "ID=" + obj.ID + "&Del=1" };

ajaxObj.getData(function (data) {

initHtml();

}, function () {

// Error

});

}

}

];

* 页码展示区

1. 页面发送一个异步请求
2. 定义一个ajaxObj对象

var ajaxObj = CommonFun\_Ajax();

1. 设置异步请求方式

\*type

表示：请求方式"get"、"post"

ajaxObj.url = "get”

默认请求方式：get

1. 设置请求地址

\*url

表示：请求地址

例如：

ajaxObj.url = "../api/A01\_Login/Login";

\*data

1. 设置请求数据

dataObj.Fun\_Data\_Add("ID", "@ViewBag.UserID");

1. 设置请求数据

表示：发送至请求地址的数据

例如：

ajaxObj.data = { "para": dataObj.Fun\_Serialize\_All() };

1. 页面数据序列化

\*Fun\_Serialize\_All();

表示：将检查项序列化为url。

1. 发送异步请求

\*getData(fun\_success, fun\_error)

表示：发出请求

例如：

ajaxObj.getData(function (data) {

if (data.ret\_status == ParaData.ret.Succes) {

CommonFun\_Href("/A02\_HomePage/Home");

} else {

alert("用户名、密码错误！");

}

}, function () {

// Error

})

1. 页面跳转

CommonFun\_Href("/A02\_HomePage/Home");

function CommonFun\_Href(url) {

var index = url.indexOf("?");

if (index >= 0) {

url += "&r=" + Math.random();

} else {

url += "?r=" + Math.random();

}

window.location.href = url;

}

1. 弹出页

举例：修改密码

CommonFun\_OpenLayer({

"Type": "700\*550"

, "Title": "重置密码"

, "Url": "/A02\_HomePage/EditPassword"

, "Fun\_End": function () { } });

实现细节

function CommonFun\_OpenLayer(obj) {

var para = "";

if (obj.Data != null) {

$.each(obj.Data, function (val1, val2) {

para += "&" + val1 + "=" + val2;

});

}

var index = layer.open({

type: 2,

title: obj.Title,

area: ["700px", "550px"],

fixed: true, //不固定

maxmin: true,

offset: "auto",

content: obj.Url + "?r=" + Math.random() + para,

end: function () {

obj.Fun\_End();

}

});

layer.iframeAuto(index);

}

1. 弹出等待页面

layer.load(2);

1. 关闭等待页面

layer.closeAll("loading");

1. 关闭弹出页

parent.layer.close(parent.layer.getFrameIndex(window.name));

1. 新增/编辑页面

举例：修改密码

<div style="margin:20px;">

<div class="form-group" style="height:280px;">

<div class="input-group">

<div class="input-group-addon" style="width: 100px; text-align: right">密码：</div>

<input type="password" style="width:480px;" class="form-control" id="Password" maxlength="20">

</div>

<div style="height:5px;">&nbsp;</div>

<div class="input-group">

<div class="input-group-addon" style="width: 100px; text-align: right">确认密码：</div>

<input type="password" style="width:480px;" class="form-control" id="Password2" maxlength="20">

</div>

</div>

<div class="form-group" style="text-align:right;">

<button type="button" class="btn btn-primary" onclick="Do\_Save();">提交</button>

</div>

</div>

1. 后台数据处理

举例：用户登录

public class A01\_LoginController : ApiController

{

private Model\_Ret \_model\_ret = new Model\_Ret();

[HttpGet]

public string Login(string para) {

para = HttpUtility.UrlDecode(HttpUtility.UrlDecode(para, Encoding.UTF8), Encoding.UTF8);

T1\_User obj = new T1\_User();

MyClass<T1\_User> myClass = new MyClass<T1\_User>(ref obj, para);

obj.Password\_MD5 = MD5.Encode(obj.Password);

DataTable dt = new DataTable();

\_model\_ret.ret\_status = obj.Login\_GetOne\_Limit(ref dt);

if (\_model\_ret.ret\_status == (int)MyEnum.Enum\_Ret.Succes) {

CookieHelper.SetCookie("UserID", dt.Rows[0]["ID"].ToString());

}

return \_model\_ret.Get\_Ret();

}

}

1. 请求数据解码
2. 请求数据绑定到类
3. 查询数据
4. 更新数据
5. 返回前台
6. HashTable
7. PropertyInfo
8. 首页页面结构

<!--Header-part-->

<div id="header">

<h1><a href="javascript:void(0);">红岭安全生产系统</a></h1>

</div>

<!--close-Header-part-->

<!--top-Header-menu-->

<div id="user-nav" class="myNavbar myNavbar-inverse">

<ul class="myNav">

<li class="dropdown" id="profile-messages">

<a title="" href="javascript:void(0);" data-toggle="dropdown" data-target="#profile-messages" class="dropdown-toggle" onclick="Fun\_ShowMenu()">

<i class="icon icon-user"></i>&nbsp;

<span class="text">欢迎你，@ViewBag.UserName</span>&nbsp;

<b class="caret"></b>

</a>

<ul class="myDropdown-menu">

<li><a href="javascript:void(0);" onclick="Fun\_EditPassword();"><i class="icon-user"></i> 修改密码</a></li>

</ul>

</li>

<li class=""><a href="javascript:void(0);" onclick="Fun\_LoginOut();"><i class="icon icon-share-alt"></i> <span class="text">&nbsp;退出系统</span></a></li>

</ul>

</div>

<!--close-top-Header-menu-->

<!--sidebar-menu-->

<div id="**sidebar**" style="OVERFLOW-Y: auto; OVERFLOW-X:hidden;">

<ul></ul>

</div>

<!--sidebar-menu-->

<!--main-container-part-->

<div id="**content**">

<!--breadcrumbs-->

<div id="**content-header**">

<div id="breadcrumb"></div>

</div>

<!--End-breadcrumbs-->

<div id="**content-body**" style="overflow:auto;">

@RenderBody()

</div>

</div>

<!--end-main-container-part-->

1. 控制器基类BaseController

控制器基类BaseController继承Controller，并且重写OnActionExecuting方法，该方法在控制器action方法被调用之前执行。

重写OnActionExecuting方法主要逻辑如下：

* 获取请求（request）的查询字符串中的” CurrPage”参数值，并赋值给ViewBag对象
* 获取请求（request）中名为“UserID”的Cookie值，判断是否存在该Cookie值，如果不存在该cookie，则跳转至登录页面。
* 根据UerID查询用户ID、Name、所属组织编码、角色编码、角色类型等信息，并将这些信息设置至ViewBag对象中。如果找不到UserID的上述信息，则跳转至登录页面。
* 并将UserID用户对应的角色对应对的**菜单**赋值给ViewBag对象MyPages

1. 缓存帮助类CacheHelper

缓存帮助类中定义了几个静态方法。

* public static object **GetCache**(string **cacheKey**)

1. 首页菜单

**菜单对象menus**

* Data： 菜单项数据，对象数组，对象结构如下：{Code，Title，Children}，其中Children，也是一个对象数组，对象元素结构如下：{ Code ，Title，Url }。Data菜单项数据的来源ViewBag对象MyPages属性。
* CurrPage：当前页面，数据来源ViewBag对象CurrPage属性
* fun\_dataInit：根据ViewBag对象MyPages属性，初始化Data菜单项数据。
* fun\_htmlInit：生成菜单Html。生成逻辑如下：
* 遍历数组Data，为数组中的每个元素生成一个<a>标签（**一级菜单**）。
* 遍历一级菜单下的Children数组（**二级菜单**），通过Code判断当前页面是否是正在遍历的二级菜单，如果是，则设置相应的选中属性。为每个二级菜单项，添加点击事件，调用fun\_click方法
* 如果当前页面是，正在遍历的一级菜单的第一个二级菜单，则将一级菜单展开。
* 将生成的html字符串，赋值给页面菜单容器
* fun\_click：菜单点击事件处理方法，该方法有两个参数，code，表示点击的菜单项的code，url，表示跳转的页面。

1. 列表页面
2. 页面对象

列表页面对象PageListFun\_Data，对象成员如下：

* Guid
* PageList\_Index
* PageList\_Count
* UserID
* OrgCode
* RRoleCode
* DRoleType
* Para1
* Para2
* Para3
* Para4
* Para5
* Para6
* Para7
* Para8
* Para9
* PageList\_Json，将页面内容转换成json字符串。

页面中包含以Para1 – Para9命名的Html标签元素，这里先取得这些元素的值，再把这些值拼接成url字符串形如：para1=value1&para2=value2……

* PageList\_Data
* PageList\_RowCount
* PageList\_Num
* PageList\_BtnInfo，一个对象，包括：Data，数组；BtnI，数字；Btns，数组
* Fun\_Loading
* Fun\_InitData，页面数据初始化方法。

function (data)

* 如果data是空，如果页面上，当前列表页面的页下标（this.PageList\_Index）大于1，则，将页下标减一，调用Fun\_Loading（）方法。；否则，即当前页面的页下标不大于1（就是1），将对象PageList\_Data置空，将对象PageList\_RowCount置1。
* 否则，参数data不为空，将参数data赋值给对象PageList\_Data。将参数data作为数组访问，将参数data数组中的每个元素作为对象访问，如果，data数组第一个元素对象的c属性不为空，且大于0，将该值赋值给对象的PageList\_RowCoun属性，否则将PageList\_RowCoun属性设置为1.
* 调用Fun\_ResetList方法
* Fun\_ResetList，列表重置方法。

function ()

* 如果页面数据行数，PageList\_RowCount，小于1条，设置页码数组PageList\_Num，为空数组。
* 如果页面数据行数，PageList\_RowCount，大于等于1条，首先取得最大页面值。
* 判断当前页码是否大于最大页面，如果大于最大页面，则将当前页码设置为最大页码。
* 定义一个数组，将当前页码之前的五个页码（页码值大于0）压入数组中。
* 将当前页码及当前页码后五页的页码（不大于最大页码）压入数组。
* 最后将这个定义的数组赋值给对象的PageList\_Num属性。
* Fun\_InitDiv，根据页码数组，生成页码html，并插入页面html
* Fun\_Bind\_Div：页码html绑定点击页面跳转操作
* Fun\_GoTo：页面跳转方法
* 计算最大页数
* 设置正确的页码值
* 调用Fun\_Loading，重新加载数据

1. 员工新增Thu
2. 员工列表Wen
3. 员工编辑\删除Tus
4. 员工导入Fri、Sat