EMP 参考手册

前端页面组件开发手册

Reference

For Version2.2

目录

版权说明

本文件系 EMP 前端页面组件的相关说明文档。文档中出现的任何文字叙述、文档格式、插图、照片、方法、过程等内容，除另有特别注明，版权均属北京宇信易诚科技有限公司所有，受到有关产权及版权法保护。任何个人、机构未经北京宇信易诚科技有限公司的书面授权许可，不得复制或引用本文件的任何片断，无论是通过电子形式或非电子形式。

文档信息

文档版本编号:	1.2	文档版本日期:	2009 年 02 月 13 日
起草人:	刘必强	起草日期:	2008 年 11 月 22 日

版本记录

版本编号	版本日期	创建/修改	说明
0.1	2008-11-22	刘必强	创建提纲
1.0	2008-12-25	刘必强	完成文档
1.1	2009-02-09	刘必强	更新文档，增加对表单标签、操作按钮组件和 JS 工具类的介绍
1.2	2009-02-13	刘必强	更新文档，修改表单标签的介绍，增加 page 标签的介绍

1. 简介

1.1. 目的

本文档详细描述了 EMP 前端页面组件的原理、功能以及相应的扩展机制。在此基础上，面向**应用客户**，对应用平台功能使用提供的一份详细说明文档。

使用说明主要满足以下三个方面的要求：

1. 系统运行环境要求和说明。
2. 系统安装及初始化应用说明。
3. 系统核心功能组件功能说明和配置说明。

1.2. 摘要

本文档主要从以下几个方面进行详细阐述：

1. 运行平台环境要求和说明
2. 系统安装及初始化说明
3. 核心组件功能及配置说明

1.3. 引用前提

N/A

1.4. 参考文档

文档名称	文件名称	版本号	备注

1. 标签对象概述

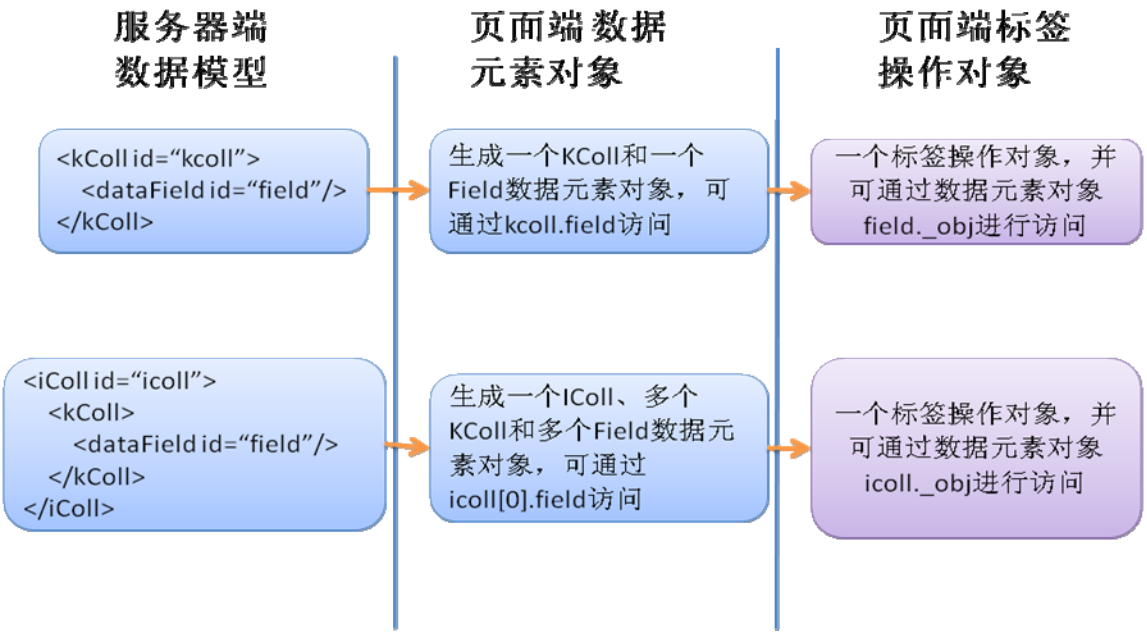
对于 EMP 的大部分标签而言，都会在页面端生成相应的 HTML 代码并将这些 HTML 代码“**注册**”成一个个的 JS 对象。这样的一个好处是可以通过这些 JS 对象控制、操作相应的 HTML 内容，实现页面上的灵活展现。并且通过对这些 JS 对象的属性、方法进行统一的命名可以使得在页面 JS 编码时通过使用**相同的接口名称(方法名称)**实现类似的功能，这样的好处是页面编程人员可以不用考虑不同浏览器的兼容性以及不同标签的特殊性，减低对页面编程人员的要求和开发繁琐性。

页面端所生成的标签对象主要分为两种：

一种称为**标签操作对象**，该对象是与 EMP 标签是紧密关联的，通过该对象就可以直接对 EMP 标签所生成的页面内容进行控制。

另外一种称为**数据元素对象**，该对象是参照于 EMP 服务器端的数据模型的结构进行设计的。EMP 的数据模型主要包括三种：IndexedCollection、KeyedCollection、DataField，相应的在页面端也设计了三种数据元素对象：IColl、KColl、Field。

数据元素对象与标签操作对象有着特定的关联关系：根据标签操作对象所要展现的服务器端的数据模型生成页面端的数据元素对象。如标签操作对象要展现的是 aColl.bField 数据，该数据在服务器端的数据模型为一个名称为 aColl 的 KeyedCollection 中包含着 bField 数据域，那么在页面端则会生成一个名称为 aColl 的 KColl 对象以及一个 bField 的 Field 对象，同时将 Field 对象作为 KColl 对象中的一个属性进行访问，即得到 aColl 对象之后就可以通过 aColl.bField 得到 bField 对象。当然通过 bField 对象也能够访问到标签操作对象并通过标签操作对象对页面内容进行操作。也就是说**数据元素对象更符合 EMP 的数据模型结构并且可以在页面端直接调用，同时数据元素对象没有实际的功能实现，需要通过标签操作对象间接的对标签的相关属性进行操作。另外标签操作对象无法在页面端直接调用，需要通过数据元素对象才能被页面端所访问**



需要注意的是：在实际开发过程中，如果需要标签自动的完成注册的工作，那么就**需要使用 emp:page 标签将其它所有的 EMP 标签包含进来**。

2. 标签操作对象

2.1. 概述

标签操作对象顾名思义是对页面的 HTML 标签的控制和操作的对象。基本上每个 EMP 标签都会在浏览器端生成一个标签操作对象(一些与后台数据之间无直接关联的标签除外，如 emp:url 只生成一个 url 的字符串、emp:label 只生成一些显示的字符串等等)，或者说每个标签操作对象都关联于一个 EMP 标签。这些根据 EMP 标签生成的标签操作对象可以对该对象所对应的标签进行操作，如隐藏、取值、设值等等。

2.1.1. 目的

对于普通的 HTML 标签而言，如果在页面上需要通过 JS 进行访问的话，只能通过 document.getElementById 或 document.getElementsByName 进行访问，同时由于对于不同类型的 HTML 标签，如果想要实现相似的功能可能所使用的 JS 方式也不太一样，这样一来就要求开发人员必须对 HTML、JavaScript 比较了解才能实现较为复杂的页面显示交互(单纯的页面内部的显示交互)。而通过将普通的 HTML 标签封装成一个标签操作对象，那么就可以通过标签操作对象提供给开发人员一个统一的操作接口。例如可以通过标签操作对象的 _renderReadonly(true)方法将该标签操作对象所对应的标签进行隐藏，在这过程中开发人员可以不用去关心不同的 HTML 标签对于隐藏的不同处理方式，可以降低对开发人员的要求，提高开发效率。

除了用于提供统一的接口外，还有一个目的是为了标签操作的一体化，简单的讲是通过对一个标签对象的操作可以使得与标签对象所对应的标签以及其周边相关联的标签实现统一的控制、操作。例如一个输入框，其对应的标签操作对象在使用_renderReadonly(true)对标签进行隐藏时，不仅仅要隐藏输入框(最基本的部分)，如果该输入框还包含了标签前后的说明，那么也需要将这些进行隐藏。更进一步来讲，当前的输入框隐藏后，也需要使得后面的标签进行相应的移动、扩展而使得页面显示更加的人性化。这些功能如果没有进行相应的封装，那么将会导致页面上的 JS 代码非常的冗杂，而且无法保证质量。

2.1.2. 实现原理

在介绍实现原理之前，首先需要介绍的是所谓的“注册”机制。由于浏览器只支持标准的 HTML 代码(虽然各个浏览器在支持上略有不同，但大体上能够保证对标准 HTML 的支持)，那么为了使特定的标签能够生成指定的标签操作对象，那么就必须“人为”的将相应的标签“注册”成标签操作对象。由于浏览器对于某些标签的属性要求比较严格，因此为了不引起浏览器解析混乱，在生成指定的标签时都使用一个 SPAN\DIV 标签来设置标签操作

对象的相应属性，然后在 SPAN\DIV 标签中才是真正所需要生成的 HTML 标签，如：

```
<span id='emp_field_User.username' title='员工姓名' type='Text' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' required='true' colSpan='1' readonly='false' onlyControlElement='false' value='' maxlength='45' rendered='false'>
    <input name='User.username' value='' maxlength='45' class='emp_field_text_input'/>
    <span class='emp_field_required'>*</span>
</span>
```

对于一个 INPUT 标签而言，其属性是有限制的。但为了使其所注册的标签操作对象能够拥有相当丰富的属性，因此在 INPUT 标签外围定义了一个 SPAN\DIV 标签，然后将标签操作对象的相应属性定义成 SPAN\DIV 标签的属性。

不过，在一个页面中，可能存在着众多的 SPAN\DIV 标签，有些是由 EMP 标签库生成的，而有些可能是编辑页面时手工编写的。那么就需要明确的定义哪些 SPAN\DIV 是对应着一个标签操作对象。因此，EMP 标签库在生成相应的 SPAN\DIV 标签时，还需要显式的定义相应的 SPAN\DIV 是一个标签操作对象。其具体的方式是将 SPAN\DIV 标签所对应的 ID 设置到页面的 page.objectsDefine 对象中(其中的 page 是一个 EMP.util.Page 对象，而且是一个页面的保留字，即在页面开发中，不能使用 page 来表示其它的变象)。如：

```
if(!page){
    var page = new EMP.util.Page();
}
page.objectsDefine = {
    dataFields : ['User.username'],
    dataTables : [],
    relatedTabs : [],
    relatedSelects : [],
    dataDics : {}
};
page.renderEmpObjects();
```

如此，浏览器在载入该页面时，就会根据 page 对象找到相应的 SPAN\DIV 标签进行解析，并注册成一个标签操作对象。

在实现开发过程中，上述的注册操作都已经由 EMP 标签库自动完成。但是如果想要不使用 EMP 标签库的同时又需要将某个标签注册成标签操作对象，那么就可以通过手工编写的方式确定 page.objectsDefine 对象的内容。具体的方式可参见相应的扩展实现部分。

开发人员需要注意的是：

1、 引入 EMP 的标签库：emp.tld

2、 使用 emp:page 标签将页面中所有需要注册成标签操作对象的 EMP 标签包容起来，通常的做法

是在<html>标签的外围使用<emp:page>标签

3、 在页面中，page 是一个保留字(如同在 JS 中也存在着众多的保留字)，不能使用 page 来表示其它变量

4、 页面中的 EMP 标签必须保证 ID 惟一性，并且从原则上 EMP 标签及 EMP 标签生成的 HTML 标签与其它手工编写的 HTML 标签也需要保证 ID 惟一性(如上面例子中的 User.username 以及 emp_field_User.username，这两者名称必须是惟一的)

2.2. 单数据标签操作对象

2.2.1. 基本原理

所谓单数据标签操作对象指的是页面上一个相对独立的标签的标签操作对象(**单个数据是与列表数据标签操作对象、分组数据标签操作对象进行区分的，同时列表数据标签操作对象与分组数据标签操作对象中都包含着单数据标签操作对象**)。由前一章节可知，每个 EMP 标签都会生成特定的 SPAN/DIV 标签，那么对于单数据标签操作对象所对应的 EMP 标签而言，其生成的是特定的 SPAN 标签(在这里将与单数据标签操作对象相关联的 EMP 标签称之为 EMP 单数据标签，相应的其它标签称为 EMP 列表数据标签和 EMP 分组数据标签)，如下图如示：

```
<span id='emp_field_User.username' title='员工姓名' type='Text' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' required='true' colSpan='1' readonly='false' onlyControlElement='false' value='' maxlength='45' rendered='false'>
    <input name='User.username' value='' maxlength='45' class='emp_field_text_input'/>
    <span class='emp_field_required'>*</span>
</span>
```

在生成的 SPAN 标签中，有最关键的一个属性：type。type 属性用于表示该 SPAN 标签所对应的单数据标签操作对象的类型，而类型又对应于标签操作对象的类名：“EMP.field.” +type 属性值。如上表中所对应的标签操作对象的类名就是 EMP.field.Text。对于该标签对象实现类的相关信息就可以通过相应的 JS 文件进行查看(对于单数据标签操作对象，实现类的代码都在 fields.js 文件中，其中每一模块都代表着一个实现类)。

其中的实现类的叫法使用的是 JAVA 中的概念，实际上实现类也是一个对象，也能直接进行操作，但为了使开发人员更容易理解，在这里使用了 JAVA 中的实现类的概念。

通过实例化一个实现类，在浏览器端就生成了一个单数据标签操作对象，然后再遍历 SPAN 标签中的属性列表作为单数据标签操作对象中 config 属性列表的内容，最后再由各个实现类对 config 属性列表中的内容进行处理。**其中，为了兼容不同的浏览器，在设置 config 属性列表中，将所有的属性都转成小写字母(在 FireFox 浏览器中，不管属性中是否有大写字母，在分析 SPAN 标签的属性列表时得到的都是小写字母的属性名称)。**

除了要生成相应的 SPAN 标签用于解析、生成单数据标签操作对象外，还需要在 page.objectsDefine 对象中的 dataFields 属性里显式的定义需要注册成单数据标签操作对象的名称。如下图所示：

```
if(!page){
	var page = new EMP.util.Page();
}
page.objectsDefine = {
	dataFields : ['User.username'],
	dataTables : [],
	relatedTabs : [],
	relatedSelects : [],
	dataDics : {}
};
page.renderEmpObjects();
```

如此，浏览器在载入该页面时(执行 page.renderEmpObjects 方法)，就会根据 page.objectsDefine 中的 dataFields 对象找到与 User.username 相对应的 SPAN 标签(对应的方式就是在页面中查找 ID 为“emp_field_”+page.objectsDefine 中定义的对象名称)进行解析，并注册成一个单数据标签操作对象。

2.2.2. 对象的控制范围

对于单数据标签对象而言，其能够操作的范围至少包括了其对应的 SPAN 标签。但正如一开始所说的标签操作对象的第二个目的：为了标签操作的一体化，单数据标签对象在特定的情况下还可以对所对应的标签周边相关联的 HTML 标签实现统一的控制、操作。

这种特定的情况指的是生成的页面 HTML 源代码的结构(或者说生成的标签与其周边相关的 HTML 标签的结构)如下表如示：

```
<td>
	员工姓名
</td>
<td>
	单数据标签操作对象对应的 HTML 标签以及其它的 HTML 标签
</td>
```

只要能够保证页面的结构如上表如示，那么单数据标签操作对象在进行隐藏、显示操作时就可以将两个 TD 标签同时隐藏、显示，同时还能够对该结构前后相同结构的单数据标签操作对象进行调整，例如改变 TD 的 colSpan 属性，使得后面的单数据标签操作对象能够扩

充成一整行进行显示等等。这是将特定标签注册成单数据标签操作对象的一个优势，即整个页面中的标签都通过注册进行管理、显示。

在大部分应用中，EMP 单数据标签都是配合着 EMP 的分组标签使用的，而 EMP 分组标签会自动的生成上表中的结构(EMP 分组标签的详细介绍请见相关章节)。而如果在手工编写页面中无法构成上表中的结构，那么必须确保单数据标签操作对象只能操作当前的 SPAN 标签。对于这种情况可以设置 EMP 单数据标签的 onlyControlElement 属性为 true 来表示，在这种情况下，如果要实现类似一体化的功能，那么只能在特定的页面中通过多个的 JS 操作进行实现。

2.2.3. 对象的基本属性和方法

2.2.3.1. 基本属性

- tag：单数据标签操作对象所对应的 span 标签
- data：单数据标签操作对象所对应的 Field 数据元素对象(见具体的 Field 数据元素对象介绍)
- dataName：相应的 EMP 单数据标签所要展现的数据域名称(当提交表单时，还是使用该属性)
- id：span 标签中的 id 属性
- title：单数据标签操作对象的描述(通常是中文描述信息)
- value：单数据标签操作对象的值
- initValue：单数据标签操作对象的初始值(即页面载入时的取值，该值通常用于重置时使用)
- checkMessage：单数据标签操作对象的校验结果信息，该属性通常用于业务层次的校验使用(具体见业务校验章节)
- table：是否包含于列表标签操作对象，如果是的话，则该属性为列表标签操作对象
- config：所有其它可以通过 SPAN 标签定义的属性列表
- checkNext：校验失败后是否继续校验后面的对象。由于默认的失败信息是在对应的 HTML 标签后面添加错误信息，可以同时显示多个域的错误信息，因此该值为 true。若失败信息使用 alert 方式弹出，则为了避免多个对象出错时不断弹出对话框，需要设置为 false。
- container：用于一体化控制的容器，缺省情况下使用的是 EMP.widget.DataFieldContainer，在该对象中拥有 parentTD 和 labelTD，可以对单数据标签操作对象所相应的 TD 以及标题描述的 TD 进行控制。如果复杂的页面结构也要使用类似的一体化，则可以重新定义一个 container 实现类(具体见容器类的扩展章节)。

2.2.3.2. 基本方法

- _parseParams：对 SPAN 标签进行解析并注册成单数据标签操作对象的相关属性(主要是将所有

标签属性注册到对象的 config 属性中)。如果其中定义了 dataType 属性，则会自动的对 SPAN 标签设置“emp_datatype_”+your dataType 的样式

- _initialize：对当前单数据标签操作对象的初始化，在 SPAN 标签解析结束后进行调用。该方法通常由具体的实现类进行重载。
- _renderStatus：处理单数据标签操作对象的相关**显示状态**。无参数，由对象的 config 属性中的 hidden、disabled、readonly 和 required 属性进行操作
- _renderHidden：处理单数据标签操作对象的隐藏状态，参数是 true\false。如果是 true，则会给当前的 SPAN 标签添加 emp_field_hidden 样式
- _renderReadonly：处理单数据标签操作对象的只读状态，参数是 true\false。如果是 true，则会给当前的 SPAN 标签添加 emp_field_readonly 样式
- _renderDisabled：处理单数据标签操作对象的有效状态，参数是 true\false。如果是 true，则会给当前的 SPAN 标签添加 emp_field_disabled 样式
- _renderRequired：处理单数据标签操作对象的必输状态，参数是 true\false。如果是 true，则会自动的在 SPAN 标签中添加一个 SPAN 标签用于显示”*”号
- setValue：对单数据标签操作对象进行设值并在页面上进行展现(在展现前可能进行了转化，如使用了 dataType、修饰字典等)
- getValue：根据单数据标签操作对象在页面上的展现得到其**真实值**
- toForm：将单数据标签操作对象的**真实值**复制到指定表单。该方法通常用于提交之前将真实值复制到指定表单后直接提交表单，以保证所提交的数据都是经过校验的真实值
- doCheckAll：检查输入数据的合法性。包括了数据的必输和字符长度校验、数据类型的校验以及业务层次的校验。该方法返回 true/false
- doCheckLength：检查输入数据的必输和字符长度。该方法通常由具体的实现类进行重载，常见的重载方式是对定义 required、maxlength 和 minlength 进行判断，该方法返回 EMPTools.message(见该对象的详细介绍)
- doCheckDataType：检查输入数据是否通过 dataType 校验。该方法通常由具体的实现类进行重载，常见的重载方式是使用 EMP.type.Base 中的 check 方法进行判断，该方法返回 EMPTools.message(见该对象的详细介绍)
- doCheckBussiness：进行业务层次的校验。该方法通常不做修改，主要的实现方式是判断当前页面中是否存在着 pageBussinessCheck 名称的方法，如果存在，则表示当前页面需要进行业务层次的校验，那么就会调用 pageBussinessCheck 进行校验，其中 pageBussinessCheck 方法的参数是当前的单数据标签操作对象，返回值是 EMPTools.message。该方法返回 true\false
- reportError：显示单数据标签操作对象的错误信息，默认方式是在 HTML 元素后面添加一个用于显示错误信息的 SPAN
- clearError：清除所显示的错误信息
- focus：将焦点置于单数据标签操作对象所指定的 HTML 元素上。该方法通常由具体的实现类进行重载
- reset：对单数据标签操作对象的值进行重置，使用的就是 initValue

在业务层次的校验中，通常一开始需要判断当前正在判断的是哪个对象，然后再根据不

同的对象进行不同的校验，如下表中所示：

```
function pageBussinessCheck(obj){
    if(!obj){
        return EMPTools.message(true);
    }
    if(obj.dataName == "User.userid"){
        var value = obj.getValue();
        if(value == "guest"){
            return EMPTools.message(false, "游客没有新增的功能!");
        }
    }
    return EMPTools.message(true);
};
```

2.2.4. EMP单数据标签基本属性

EMP 单数据标签就是与单数据标签操作对象相对应的一套标签库。其中所有的 EMP 单数据标签都是继承自 com.ecc.emp.ext.tag.field.EMPExtFieldBase，因此 EMP 单数据标签拥有着一些公用的基本属性。

- id：标签所要展现的字段的惟一标识
- label：标签的显示信息，该值可以使用多语言进行处理
- defvalue：用于显示的默认值，如果要显示的值不存在或为空，则使用该默认值。缺省情况下，该值为空串
- dictname：用于修饰的修饰名称，该值表示的是一个 IndexedCollection 的名称，若当前要展现的数据模型中没有该 IndexedCollection，则在当前数据模型中找“dictColl.”+dictname 所表示的 IndexedCollection(dictColl 表示的是一个 KeyedCollection，主要用于统一的管理修饰项的缓存)
- fieldErrorDataName：输入校验失败的错误信息数据域名称。该属性通常用于服务器端校验失败后的显示，其中该错误信息数据域中的值就是校验失败的信息。如果标签定义了该属性且存在着数据的值，那么在页面载入时会在该标签的后面显示校验失败的信息(与页面端 JS 校验失败后显示信息是一样的方式)

  服务器端校验失误的错误信息是放在一个名称为_InputErrorMsg(EMPConstance.*ERROR_MSG_KCOLL*) 的 KeyedCollection 中，fieldErrorDataName 定义的就是这个 KeyedCollection 中的某一个数据域的名称
- languageResouce：是否需要按照多语言的规则显示，该属性由具体的标签实现决定，缺省不需要。该属性与 label 的显示没有关系，label 都会调用多语言进行显示，该属性通常用于表示标签

的主体(如 INPUT 等)是否需要使用多语言支持

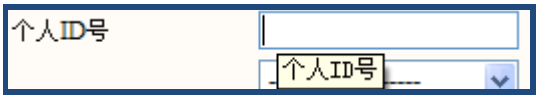
- cssClass：整个标签(span)的样式名称(通常用于定义其中特定元素样式)，缺省值是 emp_field
- cssRequiredClass：必输标志符号的样式(即*号的样式)，缺省值是 emp_field_required
- cssErrorClass：校验出错信息的样式，缺省值是 emp_field_error
- cssLabelClass：标签的显示名称的样式(只适用于布局组件中)，缺省值是 emp_field_label
- cssFlatClass：标签只以文本的方式显示数值时的样式，缺省值是 emp_field_flat
- cssElementClass：标签内具体元素的样式，没有缺省值，由各个具体的标签决定
- cssTDClass：标签的数据部分在布局组件和表格组件中的 td 属性，缺省值是 emp_field_td
- tabindex：HTML 标准属性，tab 键遍历的顺序
- accesskey：HTML 标准属性，快捷键访问方式
- help：tooltip 帮助
- required：是否必输
- hidden：是否隐藏
- disabled：是否无效状态
- flat：是否只以文本方式显示数值
- readonly：是否只读
- dataType：所使用的数据类型
- colSpan：所占的列数，只在布局组件中使用，在表格组件中不起作用
- onlyControlElement：当前标签是否只针对自身的 span 进行控制(不控制周边的 td、label 等)。当需要控制时就必须要求页面的组成是一个特定的结构(该特定结构的描述可见“对象的控制范围”章节)，缺省是控制的
- onclick：HTML 标准事件，单击
- ondblclick：HTML 标准事件，双击
- onmousedown：HTML 标准事件，鼠标按下
- onmouseup：HTML 标准事件，鼠标释放
- onmouseover：HTML 标准事件，鼠标移过
- onmousemove：HTML 标准事件，鼠标移动
- onmouseout：HTML 标准事件，鼠标移走
- onkeypress：HTML 标准事件，输入字符
- onkeyup：HTML 标准事件，按键按下
- onkeydown：HTML 标准事件，按键释放

需要注意的是：上面介绍的几个 HTML 标准事件都是直接由浏览器控制的，也就是说

在单数据标签操作对象中并不对这些 HTML 标准事件进行管理，其触发的顺序以及触发所调用的处理方法都遵循着 HTML 编码的规范，由浏览器进行处理。尤其是触发的顺序与单数据标签操作对象内部所包含的事件是无关的，或者说是无序的，主要的作用是为了尽量贴近标准的 HTML 标签，使得开发人员更容易理解和接受。

2.2.5. emp:text标签

emp:text 标签是用于普通的输入框的页面显示，在页面上展现为：



2.2.5.1. 标签特有属性

- maxlength：最大输入长度
- minlength：最小输入长度
- size：HTML 标准属性，输入框的长度。该属性可能会受到 CSS 样式的影响而不起作用
- cssElementClass：输入框的样式，缺省为：emp_field_date_input
- autocomplete：HTML 标准属性
- onfocus：HTML 标准事件
- onblur：HTML 标准事件
- onchange：HTML 标准事件
- onselect：HTML 标准事件

2.2.5.2. 标签操作对象特有属性

- element：在页面上展现的 INPUT 输入框对象

2.2.5.3. 配置示例

```
<emp:text id="User.username" label="员工姓名" maxlength="45" required="true" fieldErrorDataName="errorField"/>
```

2.2.6. emp:password标签

emp:password 标签是用于密码输入框的页面显示，在页面上展现为：

2.2.6.1. 标签特有属性

- maxlength：最大输入长度
- minlength：最小输入长度
- size：HTML 标准属性，输入框的长度。该属性可能会受到 CSS 样式的影响而不起作用
- cssElementClass：密码输入框的样式，缺省为：emp_field_password_input
- autocomplete：HTML 标准属性
- onfocus：HTML 标准事件
- onblur：HTML 标准事件
- onchange：HTML 标准事件
- onselect：HTML 标准事件

2.2.6.2. 标签操作对象特有属性

- element：在页面上展现的 INPUT 输入框对象

2.2.6.3. 配置示例

```
<emp:password id="User.password" label="密码" maxlength="10" required="true" help="请输入密码"/>
```

2.2.7. emp:link标签

emp:link 标签是用于显示链接，在页面上展现为：

admin	2008-10-21	11:10:01

或者是：

查看	2008-10-21	11:10:01

2.2.7.1. 标签特有属性

- operation：链接触发的 JS 方法(只有在 href 未定义的情况下，该属性才起作用)
- href：标签所对应的链接
- reqParams：链接中需要携带的参数
- opName：链接上的显示字符串或者作为图片的 alt 属性(如果未定义，则直接显示数据的值)
- imageFile：链接上显示的图片(若未定义，则由 opName 决定链接上显示的文本)
- needLocale：对图片，是否需要使用多语言处理，在图片名后加后缀，缺省是不使用的
- cssElementClass：链接的样式，缺省为：emp_field_link_a
- cssImageClass：链接上显示图片的样式，缺省是 emp_field_link_image
- target：HTML 标准属性

其中需要特别说明的是 reqParams 属性，它表示的是链接中携带的参数，在生成页面时会将这些参数放在 url 的后面。reqParams 的定义格式就是 url 中参数的定义格式，其中以$your dataName;来表示服务器端的一个变量。如 url 属性是 showInfo.do，reqParams 属性是 userId=$currentUserId;&type=2，而在服务器端 currentUserId 的值为 admin，那么最终生成的 url 就是：showInfo.do?userId=admin&type=2

对于所有的 EMP 标签而言，url 和 reqParams 属性都遵循上面的原则

2.2.7.2. 标签操作对象特有属性

- element：链接(A 标签)的对象
- config.href：在链接上的 href 属性

2.2.7.3. 配置示例

```
<emp:link id="User.userinfo" label="个人详细信息" opName="点击查看详情..." href="showUserInfo.do"/>
```

2.2.8. emp:multilink标签

emp:multilink 标签只适用于列表情况下，其作用是将多个的 emp:link 标签放到列表中的同一列进行显示，在页面上展现为：

操作时间	操作类型	操作结果	操作
11:14:07	queryDetail	true	修改 删除
11:14:06	queryDetail	true	修改 删除

2.2.8.1. 标签特有属性

无

2.2.8.2. 标签操作对象特有属性

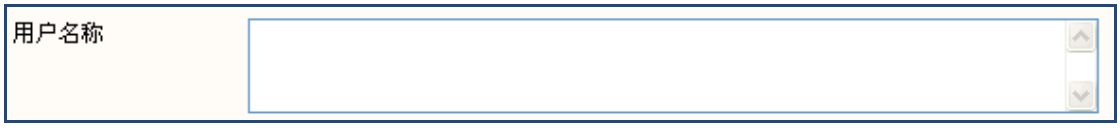
- elements：当前标签中所包含的所有链接对象(A 标签对象)的集合
- config.hrefs：链接对象集合对应的 href 属性的集合
- config.linkcount：所包含的链接的个数

2.2.8.3. 配置示例

```
<emp:multilink id="multiOpLink">
    <emp:link id="update" label="修改" opName="修改" operation="update"/>
    <emp:link id="delete" label="删除" opName="删除" operation="delete"/>
</emp:multilink>
```

2.2.9. emp:textarea标签

emp:textarea 是用于多行多列输入域的页面显示，在页面上展现为：



2.2.9.1. 标签特有属性

- maxlength：最大输入长度
- minlength：最小输入长度
- cols：HTML 标准属性
- rows：HTML 标准属性
- cssElementClass：输入域的样式，缺省为：emp_field_textarea_textarea
- onfocus：HTML 标准事件
- onblur：HTML 标准事件
- onchange：HTML 标准事件
- onselect：HTML 标准事件

2.2.9.2. 标签操作对象特有属性

- element：在页面上展现的 TEXTAREA 输入域对象

2.2.9.3. 配置示例

```
<emp:textarea id="User.remark" label="备注" maxlength="1000" required="false" colSpan="2"/>
```

因为 textarea 通常都会占用页面较大的区域，所以大部分情况下都是设置 colSpan 为 2。不过这也只是一般的页面编写经验，具体配置需要考虑当前页面的布局设计

2.2.10. emp:select标签

emp:select 标签是用于下拉框的页面显示，在页面上展现为：

2.2.10.1. 标签特有属性

- defMsg：未选择状态下，下拉框的显示信息(未选择状态下，下拉框的值为"")，缺省是"-----请选择-----"
- cssElementClass：下拉框的显示样式，缺省是 emp_field_select_select
- cssFakeInputClass：只读时的显示的输入框样式，缺省是 emp_field_select_input
- size：HTML 标准属性
- onfocus：HTML 标准事件
- onblur：HTML 标准事件
- onchange：HTML 标准事件

2.2.10.2. 标签操作对象特有属性

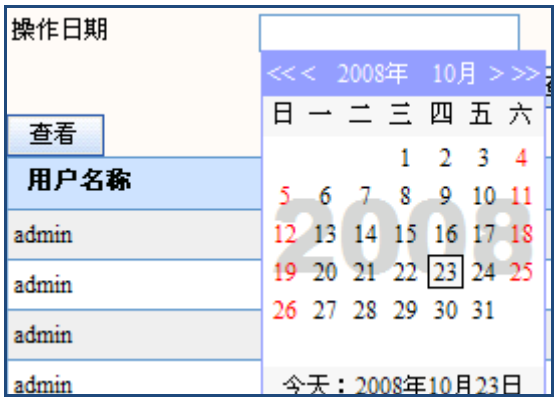
- element：在页面上展现的 SELECT 下拉框对象
- fakeinput：只读时在页面上展现的 INPUT 输入框对象
- relatedSelectDefine：专门对应于联动下拉框情况。如果当前的下拉框属于某个联动下拉框分组的一员，那么该属性值就不为空。该属性的值结构是：{group:相对应联动下拉框组件对象,next:下一个联动的单数据标签操作对象}。该属性的具体作用见“联动下拉框组件”相关介绍

2.2.10.3. 配置示例

```
<emp:select id="User.ed_career" label="全日制学历" required="false" dictname="XL" defvalue="1"/>
```

2.2.11. emp:date标签

Date 标签是在页面上显示日期以及方便日期选择的输入框，在页面上展现为：



Date 标签的大部分实现与 Text 标签一致，惟一的区别是对于 Date 标签增加了一个 click 事件，该事件的作用是弹出一个日期的选择框。同时 emp:date 标签缺省拥有名称为 Date 的 dataType 类型用于判断输入的字符是否是日期类型

2.2.11.1. 标签特有属性

见 emp:text 标签说明。其中的 cssElementClass 属性的缺省值为 emp_field_date_input

2.2.11.2. 标签操作对象特有属性

见 emp:text 标签说明

2.2.11.3. 配置示例

```
<emp:date id="User.to_duty_date" label="调任岗位起始日期" required="false" dataType="Date"/>
```

2.2.12. emp:pop标签

emp:pop 标签是一个比较特殊的标签，该标签设计的一个主要目的是为了简化选择框的实现。emp:pop 标签会在页面上生成一个请求弹出页面的按钮，通过弹出页面与当前页面之间的交互，可以从弹出页面得到相应的数据更新当前的页面。其在页面上展现为：

emp:pop 标签的大部分实现也是与 emp:text 标签一致，最重要的区别是在输入框的右边增加了一个按钮，点击该按钮之后弹出一个弹出页面

2.2.12.1. 标签特有属性

- url：弹出窗口所对应的 url 属性
- reqParams：url 中需要携带的参数
- returnMethod：弹出窗口返回时需要调用的父窗口中的方法名称(只是方法名称，不带参数定义)
- popParam：弹出窗口的属性，如窗口位置、宽度等
- dataMapping：弹出窗口返回的数据与父窗口数据之间的映射关系(定义了映射关系后会自动的对父窗口中的数据进行设值)
- cssElementClass：输入框的样式，缺省为：emp_field_pop_input
- cssButtonClass：按钮的显示样式，缺省是 emp_field_pop_button

2.2.12.2. 标签操作对象特有属性

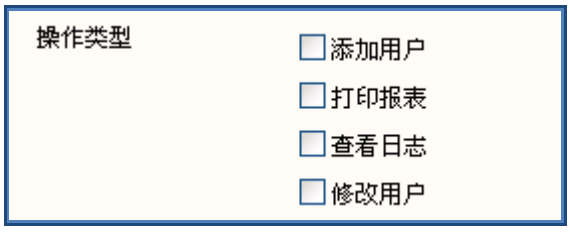
- element：在页面上展现的 INPUT 输入框对象
- button：在页面上展现的 BUTTON 按钮对象

2.2.12.3. 配置示例

```
<emp:pop id="User.org" label="所属机构" url="queryOrgDetail.do" reqParams="orgId=$currentOrgId;&type=2" returnMethod="changePageValue"/>
```

2.2.13. emp:checkbox标签

emp:checkbox 标签是在页面上显示复选框，在页面上展现为：



2.2.13.1. 标签特有属性

- checkValue：选中时的取值，如果当前值与选中值相等，则复选框显示为选中状态
- layout：是否使用 table 对多个的复选框进行分层显示
- delimiter：如果不分层显示，则定义每个复选框之间的分割符
- valueCollection：当 checkBox 对应的值为 IndexedCollection 时，该 IndexedCollection 的名称，若未定义该属性，则表示 checkBox 的值是以";"分割的字符串
- cssElementClass：复选框的样式，缺省为：emp_field_checkbox_input
- onfocus：HTML 标准事件
- onblur：HTML 标准事件
- onchange：HTML 标准事件

2.2.13.2. 标签操作对象特有属性

- elements：在页面上展现的所有复选框对象的集合

2.2.13.3. 配置示例

```
<emp:checkbox id="User.ed_career" label="全日制学历" required="false" dictname="XL" defvalue="1" checkValue="1"/>
```

2.2.14. emp:radio标签

emp:radio 标签是在页面上显示单选框，在页面上展现为：

2.2.14.1. 标签特有属性

- checkValue：选中时的取值，如果当前值与选中值相等，则单选框显示为选中状态
- layout：是否使用 table 对多个的单选框进行分层显示

- delimiter：如果不分层显示，则定义每个单选框之间的分割符
- cssElementClass：单选框的样式，缺省为：emp_field_radio_input
- onfocus：HTML 标准事件
- onblur：HTML 标准事件
- onchange：HTML 标准事件

2.2.14.2. 标签操作对象特有属性

- elements：在页面上展现的所有复选框对象的集合

2.2.14.3. 配置示例

```
<emp:radio id="User.ed_career" label="全日制学历" required="false" dictname="XL" defvalue="1" checkValue="1"/>
```

2.2.15. emp:appendPin标签

emp:appendPin 标签是在页面上显示一个密码验证框，其中主要是由两部分组成：一个是密码输入框；另一个是随机码的图片。在页面上展现为：

2.2.15.1. 标签特有属性

- codeLength：验证码长度
- imgSrc：验证码图片的来源(缺省是 dynPassword.do?cmd=verifyPin)
- altStr：验证码图片的 alt 属性
- width：验证码图片的宽度
- height：验证码图片的高度
- cssElementClass：输入框的样式，缺省为：emp_field_appendpin_input
- cssImageClass：验证码图片的显示样式，缺省是 emp_field_appendpin_image
- onfocus：HTML 标准事件

- onblur：HTML 标准事件
- onchange：HTML 标准事件
- onselect：HTML 标准事件
- autocomplete：HTML 标准属性
- size：HTML 标签属性

在这里需要注意的一点是，为了能够得到验证码的图片，在缺省情况下系统中必须存在着 dynPassword 名称的 MVC 定义，通常是定义在 empServletContext.xml 中。该 MVC 的定义比较特殊，在配置示例中有相应的例子。

2.2.15.2. 标签操作对象特有属性

- element：在页面上展现的 INPUT 输入框对象
- image：在页面上展现的验证码图片对象

2.2.15.3. 配置示例

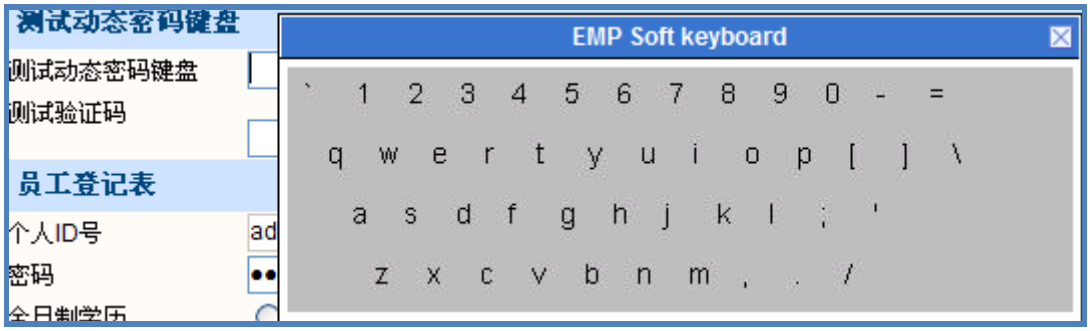
```
<emp:appendPin id="TestSoft.test22" label="测试验证码" required="false" codeLength="4" width="10" height="3" />
```

下表是必要的 MVC 定义：

```
<action id="dynPassword" class="com.ecc.emp.web.dynpassword.DynamicPasswordController"/>
```

2.2.16. emp:dynPassword标签

emp:dynPassword 标签是在页面上显示一个动态的密码键盘用于密码的输入，其主要的目的是为了防止类似于键盘监视程序获得用户输入的密码。其结果与键盘输入密码没有区别，在页面端并没有进行相应的加密行为，传送到服务器的依然是明文的密码。在点击相应的输入框时，页面上会自动的弹出一个密码键盘输入区域，在页面上展现为：



2.2.16.1. 标签特有属性

- keyBoardWidth：密码键盘宽度，缺省 400px
- keyBoardHeight：密码键盘高度，缺省 120px
- keyboardType：密码键盘类型：FULL 表示全键盘，其它则表示数字键盘
- keyBoardTitle：密码键盘的显示标题，缺省是 EMP Soft keyboard
- encodeKey：是否需要将动态键盘对照表缓存在 session 中，缺省不缓存
- imgSrc：获得动态键盘图片显示的请求，缺省是 dynPassword.do
- maxlength：最大输入长度(包括了双字节的中文等)
- minlength：最小输入长度(包括了双字节的中文等)
- cssElementClass：输入框的样式，缺省为：emp_field_dynPassword_input
- size：HTML 标准属性
- onfocus：HTML 标准事件
- onblur：HTML 标准事件
- onchange：HTML 标准事件
- onselect：HTML 标准事件
- autocomplete：HTML 标准属性

需要注意的是，与 emp:appendPin 标签一样，emp:dynPassword 标签也需要特定的 MVC 定义配合使用才能正常的显示。两个标签可以使用同一个 MVC 定义，缺省的配置可见 emp:appendPin 标签中的配置示例。

2.2.16.2. 标签操作对象特有属性

- element：在页面上展现的 INPUT 输入框对象
- keyBoard：密码键盘对象，使用的是 EMP.widget.SoftKeyboard

2.2.16.3. 配置示例

```
<emp:dynPassword id="TestSoft.test1" label="测试动态密码键盘" required="true" keyboardType="FULL" keyBoardTitle="全密码键盘"/>
```

2.2.17. emp:textSpace标签

emp:textSpace 标签是用于查询页面中表示区间查询的标签，在页面上展现为：

2.2.17.1. 标签特有属性

emp:textSpace 的相关属性与 emp:text 属性一致，详情见 emp:text 说明

2.2.17.2. 标签操作对象特有属性

- element_begin：前一个 INPUT 输入框对象
- element_end：后一个 INPUT 输入框对象

需要注意的一点是，查询标签(emp:textSpace 与 emp:dateSpace)并不对应服务器端的某个数据，更准确的是它对应于服务器端的两个数据。当提交时，查询标签是提交两个数据，名称分别是在 id 的后面加上“_begin”和“_end”，而在页面显示时并没有初始值，即该标签只做为查询使用，只适用于输入。

另外，标签操作对象的 getValue 的返回值和 setValue 的参数都不是纯粹的字符串，而是一个特定结构的对象，其结构是：{begin:"",end:""}

2.2.17.3. 配置示例

```
<emp:textspace id="Query.age" label="年龄" maxlength="3" dataType="Number"/>
```

2.2.18. emp:dateSpace标签

emp:dateSpace 标签是用于查询页面中专门用于日期区间查询的标签，在页面上展现为：

2.2.18.1. 标签特有属性

emp:dateSpace 与 emp:textSpace 标签之间的关系就跟 emp:date 与 emp:text 标签之间的关系一样，具体的属性可见 emp:textSpace 和 emp:date 标签介绍

2.2.18.2. 标签操作对象特有属性

与 emp:textSpace 标签一致，详情见 emp:textSpace 标签介绍

2.2.18.3. 配置示例

```
<emp:datespace id="Query.date" label="日期" dataType="Date"/>
```

2.3. 列表数据标签操作对象

2.3.1. 列表组成

由于列表数据标签生成的 HTML 代码比较复杂，由多个部分组成，因此在介绍列表数据标签操作对象之前先介绍一下几个组成部分。

列表数据标签生成的特定 TABLE 结构如下表如示：

```
<div id='emp_table_testTable_div'>

<table id='emp_table_testTable_table' class='emp_table' pageMode='true' selectType='1' needTableTitle='true' editable='false' rendered='false' url='/SRS/null'>

<thead>

<tr class='emp_table_title'><th id='emp_table_testTable_th_field1' columnName='field1' hidden='false'>编号</th><th id='emp_table_testTable_th_field2' columnName='field2' hidden='false'>金额</th><th id='emp_table_testTable_th_multiOpLink' columnName='multiOpLink' hidden='false'>操作</th></tr>

</thead>

<tbody id='emp_table_testTable_tbodysample' style='display:none'><tr>

<td><span id='emp_field_field1' title='编号' type='Base' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' colSpan='1' readonly='false' onlyControlElement='false' value='' rendered='false'><span class='emp_field_flat'></span></span>

</td><td><span id='emp_field_field2' title='金额' type='Base' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' colSpan='1' readonly='false' onlyControlElement='false' dataType='Currency' validateJS='EMP.type.Decimal.check(displayValue, 2, false, 100000000, -100000000, formatErrorMsg, rangeErrorMsg)' convertorJS='EMP.type.Decimal.display(realValue, 2, true, false)' rangeErrorMsg='please input [金额] between [100000000] and [-100000000]!' formatErrorMsg='input field [金额] format error,the right format as: [#,##0.00]!' value='' rendered='false'><span class='emp_field_flat'></span></span>

</td><td><span id='emp_field_multiOpLink' title='操作' type='Base' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' colSpan='1' readonly='false' onlyControlElement='false' value='' rendered='false'><a class='emp_field_link_a' href='#update'>修改</a><a class='emp_field_link_a' href='#delete'>删除</a></span>

</td></tr></tbody>

<tbody id='emp_table_testTable_tbodymain' style='display:none'>

<tr>

<td class='emp_field_td'><span id='emp_field_field1' title='编号' type='Base' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' colSpan='1' readonly='false' onlyControlElement='false' value='1' rendered='false'><span class='emp_field_flat'>1</span></span>

</td><td class='emp_field_td'><span id='emp_field_field2' title='金额' type='Base' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' colSpan='1' readonly='false' onlyControlElement='false' dataType='Currency' validateJS='EMP.type.Decimal.check(displayValue, 2,
```

false, 100000000, -100000000, formatErrorMsg, rangeErrorMsg)' convertorJS='EMP.type.Decimal.display(realValue, 2, true, false)' rangeErrorMsg='please input [金额] between [100000000] and [-100000000]!' formatErrorMsg='input field [金额] format error,the right format as: [#,##0.00]!' value='0.109374225' rendered='false'>0.109374225

</td><td class='emp_field_td'><span id='emp_field_multiOpLink' title='操作' type='Base' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' colSpan='1' readonly='false' onlyControlElement='false' value='' rendered='false'>修改删除

</td></tr>

<tr>

<td class='emp_field_td'><span id='emp_field_field1' title='编号' type='Base' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' colSpan='1' readonly='false' onlyControlElement='false' value='2' rendered='false'>2

</td><td class='emp_field_td'><span id='emp_field_field2' title='金额' type='Base' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' colSpan='1' readonly='false' onlyControlElement='false' dataType='Currency' validateJS='EMP.type.Decimal.check(displayValue, 2, false, 100000000, -100000000, formatErrorMsg, rangeErrorMsg)' convertorJS='EMP.type.Decimal.display(realValue, 2, true, false)' rangeErrorMsg='please input [金额] between [100000000] and [-100000000]!' formatErrorMsg='input field [金额] format error,the right format as: [#,##0.00]!' value='0.07940757' rendered='false'>0.07940757

</td><td class='emp_field_td'><span id='emp_field_multiOpLink' title='操作' type='Base' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' colSpan='1' readonly='false' onlyControlElement='false' value='' rendered='false'>修改删除

</td></tr>

<tr>

<td class='emp_field_td'><span id='emp_field_field1' title='编号' type='Base' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' colSpan='1' readonly='false' onlyControlElement='false' value='3' rendered='false'>3

</td><td class='emp_field_td'><span id='emp_field_field2' title='金额' type='Base' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' colSpan='1' readonly='false' onlyControlElement='false' dataType='Currency' validateJS='EMP.type.Decimal.check(displayValue, 2, false, 100000000, -100000000, formatErrorMsg, rangeErrorMsg)' convertorJS='EMP.type.Decimal.display(realValue, 2, true, false)' rangeErrorMsg='please input [金额] between [100000000] and [-100000000]!' formatErrorMsg='input field [金额] format error,the right format as: [#,##0.00]!' value='0.07145792' rendered='false'>0.07145792

</td><td class='emp_field_td'><span id='emp_field_multiOpLink' title='操作' type='Base' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' colSpan='1' readonly='false' onlyControlElement='false' value='' rendered='false'>修改删除


```
</td></tr>

<tr>

<td class='emp_field_td'><span id='emp_field_field1' title='编号' type='Base' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' colSpan='1' readonly='false' onlyControlElement='false' value='4' rendered='false'><span class='emp_field_flat'>4</span></span>

</td><td class='emp_field_td'><span id='emp_field_field2' title='金额' type='Base' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' colSpan='1' readonly='false' onlyControlElement='false' dataType='Currency' validateJS='EMP.type.Decimal.check(displayValue, 2, false, 100000000, -100000000, formatErrorMsg, rangeErrorMsg)' convertorJS='EMP.type.Decimal.display(realValue, 2, true, false)' rangeErrorMsg='please input [金额] between [100000000] and [-100000000]!' formatErrorMsg='input field [金额] format error,the right format as: [#,##0.00]!' value='0.13767034' rendered='false'><span class='emp_field_flat'>0.13767034</span></span>

</td><td class='emp_field_td'><span id='emp_field_multiOpLink' title='操作' type='Base' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' colSpan='1' readonly='false' onlyControlElement='false' value='' rendered='false'><a class='emp_field_link_a' href='#update'>修改</a><a class='emp_field_link_a' href='#delete'>删除</a></span>

</td></tr>

<tr>

<td class='emp_field_td'><span id='emp_field_field1' title='编号' type='Base' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' colSpan='1' readonly='false' onlyControlElement='false' value='5' rendered='false'><span class='emp_field_flat'>5</span></span>

</td><td class='emp_field_td'><span id='emp_field_field2' title='金额' type='Base' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' colSpan='1' readonly='false' onlyControlElement='false' dataType='Currency' validateJS='EMP.type.Decimal.check(displayValue, 2, false, 100000000, -100000000, formatErrorMsg, rangeErrorMsg)' convertorJS='EMP.type.Decimal.display(realValue, 2, true, false)' rangeErrorMsg='please input [金额] between [100000000] and [-100000000]!' formatErrorMsg='input field [金额] format error,the right format as: [#,##0.00]!' value='0.67923784' rendered='false'><span class='emp_field_flat'>0.67923784</span></span>

</td><td class='emp_field_td'><span id='emp_field_multiOpLink' title='操作' type='Base' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' colSpan='1' readonly='false' onlyControlElement='false' value='' rendered='false'><a class='emp_field_link_a' href='#update'>修改</a><a class='emp_field_link_a' href='#delete'>删除</a></span>

</td></tr>

</tbody>

<tbody id='emp_table_testTable_tbodystat' style='display:none'>

<tr><td></td><td></td><td></td></tr>

</tbody>

<tbody id='emp_table_testTable_tbodymsg' style='display:none'>

<tr><td colspan=3><span></span></td></tr></tbody>
```

```
<tbody id='emp_table_testTable_tbodypq' style='display:none'>
<tr><td colspan=3><button id='emp_pq_first'> 首 页 </button><button id='emp_pq_previous'> 上 一 页 </button>第<span id='emp_pq_currentPage'></span>页，共<span id='emp_pq_totalPage'>?</span>页 <span id='emp_pq_recordSize' style='display:none'></span><button id='emp_pq_next'> 下 一 页 </button><button id='emp_pq_last'> 尾 页 </button><input id='emp_pq_jumpInput'/><button id='emp_pq_jumpButton'> 跳 转 </button> 每 页 <input id='emp_pq_maxLine' value=''/> 行 </td></tr></tbody>
</table>
</div>
```

从上表中可以知道，EMP 列表数据标签所生成的 HTML 代码远不止一个 TABLE 标签这么简单，其中包含了**标题栏部分、数据模板部分、数据展现部分、统计部分以及最后的分页部分**。下面，将上表中的各个部分分拆进行相应的介绍：

2.3.1.1. 标题栏部分

EMP 列表数据标签根据包含在其中的 EMP 单数据标签的 label 属性组成了表格中的标题栏部分：

```
<thead>
 <tr class='emp_table_title'>
     <th id='emp_table_testTable_th_field1' columnName='field1' hidden='false'>编号</th>
     … …
 </tr>
</thead>
```

2.3.1.2. 数据模板部分

在页面开发过程中，经常会碰到一个需求：增加或减少列表中的一行。其中减少一行相对比较简单，但是在当前列表不存在任何记录的情况下增加一行就比较困难了，最主要的原因是无法确定增加的这一行的结构如何。其中一种解决方法是对每一个表格都专门写一套 JS 的处理方法，那么针对每一个具体的表格都知道相应的结构。但是这种方法有一个问题是对于每一个开发列表数据显示页面的开发人员来说，都有可能面临着这样的问题，也就都可能去写这么一套的 JS 处理方法，这就对开发人员的要求比较高。

为了解决这种问题，EMP 的列表数据标签在生成页面的 HTML 代码时，就主动的生成了一套数据模板。当新增一行时，无论当前列表是否存在着记录，列表数据标签操作对象都可以使用一个 API 对数据模板进行分析、生成相应的记录。

模板的结构如下表如示：

```
<tbody id='emp_table_testTable_tbodysample' style='display:none'>
 <tr>
    <td>
        <span id='emp_field_field1' title=' 编 号 ' type='Base' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' colSpan='1' readonly='false' onlyControlElement='false' value='' rendered='false'><span class='emp_field_flat'></span></span>
    </td>
    … …
 </tr>
</tbody>
```

如果将数据模板与数据展现部分(具体结构将在数据展现部分介绍)进行比较，会非常明显的发现两者的结构是一模一样的。数据模板其实就是代表着数据展现部分的结构，这样一来当新增一条记录时就能保证新增的记录与原有的记录保持相同的结构，可以说数据模板就是一条隐藏的没有数值的记录。

2.3.1.3. 数据展现部分

数据展现部分是整个列表数据标签操作对象的主体部分，包含着多条的记录，其中每一条记录都包含着多个的单对象数据操作对象。其结构如下表如示：

```
<tbody id='emp_table_testTable_tbodymain' style='display:none'>
 <tr>
    <td class='emp_field_td'>
        <span id='emp_field_field1' title=' 编 号 ' type='Base' class='emp_field' cssErrorClass='emp_field_error' cssRequiredClass='emp_field_required' colSpan='1' readonly='false' onlyControlElement='false' value='1' rendered='false'><span class='emp_field_flat'>1</span></span>
    </td>
    … …
 </tr>
 … …
</tbody>
```

从上表中可以看出，数据展现部分就是一个非常规整的 HTML 表格，其中每个单元格中的内容就是一个 EMP 单数据标签所生成的内容，同样的也就会“注册”成一个单数据标签操作对象，只不过这个单数据标签操作对象需要通过列表数据标签操作对象进行访问。

2.3.1.4. 统计部分

这部分的内容有待完善，故暂时不介绍

2.3.1.5. 分页部分

在实际应用过程中，列表通常都要求拥有分页的功能。尤其是作为动态的 Web 项目，符合查询条件的列表数据记录数通常都是不可预知的，那么表格的分页功能就成为了最基本的一个需求。

EMP 列表数据标签生成的分页部分的 HTML 结构如下表如示：

```
<tbody id='emp_table_testTable_tbodypq' style='display:none'>
<tr>
    <td colspan=3>
        <button id='emp_pq_first'>首页</button><button id='emp_pq_previous'>上一页</button>第<span id='emp_pq_currentPage'>1</span>页，共<span id='emp_pq_totalPage'>?</span>页<span id='emp_pq_recordSize' style='display:none'>5</span><button id='emp_pq_next'>下一页</button><button id='emp_pq_last'>尾页</button><input id='emp_pq_jumpInput'/><button id='emp_pq_jumpButton'>跳转</button>每页<input id='emp_pq_maxLine' value='10'/>行
    </td>
 </tr>
</tbody>
```

从上表中的内容就可以看出 EMP 列表数据标签已经生成了分页的主体部分，包括了总记录数、每页记录数、当前页等必要的信息。

而对于 EMP 列表数据标签如何知道这些信息呢？其原因就在于 EMP 列表数据标签会查询服务器端是否存在着一个名称为 pageInfo 的 KeyedCollection。如果存在着这么一个 KeyedCollection，那么就表示为 EMP 列表数据标签的分页信息。这个 KeyedCollection 中包含了三个数据：currentPage：表示当前是第几页；maxLine：每页能够显示的记录条数；recordSize：总的记录条数。然后列表数据标签就可以根据这些信息生成上表中的内容。

但是，对于 pageInfo 这种方式依然存在着一个局限性，那就是一个页面只能有一个列表数据标签。这是因为服务器端不能拥有多个同名的 KeyedCollection。

2.3.2. 基本原理

列表数据标签根据字面意思就是用一个特定的标签来实现对列表数据的展现。众所周

知，在 HTML 结构中，对于表格的展现通常都是通过 TABLE 标签来实现的(个别开发人员会采用其它方式配合上 CSS 样式设计来实现，这种方式比较灵活且表现丰富，但是对开发人员的要求比较高)，但是 TABLE 表格只是一个功能比较少的 HTML 元素，当开发人员需要对表格进行操作时会发现其页面的开发代码非常的繁多。例如，简单的添加一行、删除一行，再到复杂的分页显示、查询显示等等。

那么为了尽可能的降低对开发人员的要求，提高开发效率，EMP 列表数据标签就将其生成的特定 TABLE 标签“注册”为一个列表数据标签操作对象，通过对列表数据标签操作对象的控制，可以实现对列表的各种复杂的功能。而且当开发过程中，如果要实现更加复杂的功能时，可以通过个别页面开发的技术骨干在列表数据标签操作对象的基础上进行扩展 API，提供给开发人员使用。

与单数据标签操作对象一样，为了能够将相应的列表“注册”成列表数据标签操作对象，还需要在 page.objectsDefine 对象中的 dataTables 属性里显式的定义需要注册成列表数据标签操作对象的名称。如下图所示：

```
if(!page){
    var page = new EMP.util.Page();
}
page.objectsDefine = {
    dataFields : [],
    dataTables : ['testTable'],
    relatedTabs : [],
    relatedSelects : [],
    dataDics : {}
};
page.renderEmpObjects();
```

如此，浏览器在载入该页面时(执行 page.renderEmpObjects 方法)，就会根据 page.objectsDefine 中的 dataTables 对象找到与 testTable 相对应的 TABLE 标签(对应的方式就是在页面中查找 ID 为“emp_table_”+page.objectsDefine 中定义的对象名称+“_table”)进行解析，并根据前面介绍的列表的各个组成部分进行解析，注册成一个列表数据标签操作对象。在列表的解析过程中也包含了对单数据标签操作对象的解析(列表从本质上来讲是对有特定关系的单数据标签的展现)

2.3.3. 对象的基本属性和方法

2.3.3.1. 基本属性

- tag：列表数据标签操作对象所对应的 TABLE 标签
- data：列表数据标签操作对象所对应的 IColl 数据元素对象(具体见 IColl 数据元素对象说明)
- dataName：相应的 EMP 列表数据标签所要展现的列表名称(当提交表单时，还是使用该属性)
- nodata：判断服务器端是否存在所要展现的数据集合
- selectType：列表的选择类型(即点击列表中的一个记录时的选择类型)：0 代表不支持选择，1 代表单选，2 代表多选，其它选择方式待实现
- needTableTitle：是否显示标题栏
- pageMode：列表是否进行分页展现
- url：进行翻页时的默认 URL
- columnNames：列表数据标签操作对象所包含的字段的名称集合(类似于数据库表中的字段名称集合)
- recordCount：列表数据标签操作对象中所包含的记录行数
- selectedRows：各行选中标志 true\false
- tHead：标签栏部分的 tHead 元素对象
- tBodySample：数据模板部分的 tBody 元素对象
- tBodyMain：数据展现部分的 tBody 元素对象
- tBodyTotal：统计部分的 tBody 元素对象
- tBodyMsg：提示信息部分的 tBody 元素对象
- tBodyPq：分页部分的 tBody 元素对象
- msgSpan：在提示信息部分中显示提示信息内容的 SPAN 标签对象
- pageQuery：用于控制分页部分显示的分页处理对象(EMP.widget.PageQuery)
- onSelect：选中某一行时，自定义的执行方法。该属性表示的是一个方法，如果定义了该属性，则在列表中某一行被选中时自动执行该方法。该方法的参数是所选中行的行数

2.3.3.2. 基本方法

- _parseParams：对 TABLE 标签进行解析并注册成列表数据标签操作对象的相关属性，并调用 _initialize 进行初始化

- _initialize：对列表数据标签操作对象进行初始化，并为列表数据的展现添加事件、样式
- _noData：内部方法，当服务器端无数据集合定义时的处理方法
- _noRecords：内部方法，当服务器端有数据集合定义，但数据集合中无记录时的处理方法
- setMessage：设置表格的提示信息，其参数 msg 表示需要提示的信息。如果 msg 为 null，则表示隐藏提示信息，否则覆盖显示
- _getColumnNames：内部方法，从 THEAD 上取得各字段的名称。调用结束后，各字段的名称保存在属性 columnNames 中
- _regist：内部方法，对数据展现部分中所有的 EMP 单数据标签进行遍历解析，生成相应的单数据标签操作对象和 Field 数据元素对象，并将 Field 数据对象通过 KColl 数据元素对象放到当前列表数据标签操作对象的 data 列表中(这涉及到单数据标签操作对象和列表数据标签操作对象如何通过数据元素对象进行访问，这些内容将在数据元素对象部分介绍)
- _registRow：内部方法，对数据展现部分中指定一行的 EMP 单数据标签进行遍历解析，生成相应的单数据标签操作对象和 Field 数据元素对象，并将所有的 Field 数据对象放到一个 KColl 数据元素对象中返回。该方法由_regist 进行循环调用，用于列表数据的注册
- _addCss：对数据展现部分添加 CSS 样式，目前的方案是设置隔行不同颜色，分别是 row1、row2 两种样式
- _addSelectEvents：对数据展现部分添加 click 事件，用于实现鼠标点击某一行时的处理
- click：数据展现部分 click 事件所触发的方法，根据事件所触发的元素得到具体发生在哪一行，然后调用 select 方法用于表示选中该行
- select：选中某一行所调用的方法。当某一行被选中之后，添加名称为 selected 的 CSS 样式，并设置 selectedRows 属性中相应行号的值为 true
- clearAll：清除列表数据标签操作对象中的所有选中状态
- selectAll：选中列表数据标签操作对象中的所有记录行
- getSelectedIdx：API 方法，以数组形式获得当前选中记录的行号。由于 EMP 列表数据标签可以用于多选的情况，因此该方法是以数组的形式返回
- getDataValue：API 方法，以数组形式获得当前选中记录的取值集合(只包含取值，而不是单数据标签操作对象或其它对象)
- getSelectedData：以数组形式获得当前选中记录所对应的 KColl 数据元素对象(关于 KColl 数据元素对象的介绍详见相关章节)
- getParamStr：获取当前选中记录的指定字段的 GET 请求串。如果是单选情况，则 GET 请求串表示的是单个数据；如果是多选情况，则 GET 请求串表示的是列表数据。该方法有两个参数：第一个是指定的字段列表，第二个参数是前缀(即在字段名的前面需要添加的前缀，如果没有则表示不需要添加前缀)
- toForm：将整个数据展现部分的数据复制到指定表单 form 中
- _addRow：使用数据模板部分的模板往数据展现部分动态的添加一条记录
- _deleteRow：删除数据展现部分指定的一条记录，其参数是所指定的行号
- ajaxQuery：API 方法，使用指定的 url 或表单中的数据向服务器端发起请求，获得列表数据标签

操作对象的所有记录。该方法通常用于动态查询等场合，而服务器端返回的是特定结构的 JSON 字符串

- tablePageQuery：API 方法，用于列表数据动态更新的具体实现方法，通常由 ajaxQuery 方法和分页处理对象进行调用
- parseAjaxQuery：对于列表数据动态更新所发起的请求结果进行分析，刷新列表的展现

需要注意的是：列表数据标签操作对象能够直接操作的只是 TABLE 标签，也就是说列表数据标签操作对象是与 HTML 的表格是直接关联的，通过列表数据标签操作对象封装的多个 API 可以实现对表格整体的展现。由于在页面上直接操作列表的业务需求很少，因此相关的 API 也比单数据标签操作对象少，更多情况下是取得列表中的选中行的数值或指定行中指定列的值等。

2.3.4. 分页处理对象

分页处理对象是列表数据标签操作对象中的一部分，并**没有单纯的 EMP 标签与之对应**，在 EMP 列表标签中设置了分页显示之后会由列表数据标签操作对象自动创建出一个分页处理对象专门处理分页的状态与显示。这里单独拿出来介绍是为了表示分页处理的相对独立性以及 EMP 服务器端对于分页的处理模式。

对于数据库查询结果的分页通常会有三种方式：第一种是将符合查询结果的数据全部返回到页面端，然后在页面端进行分页显示；第二种是将符合查询结果的数据放在服务器端的缓存中，然后页面端根据需要从缓存中读取进行分页显示；第三种则是根据需要再从数据库中查询相应的记录。

EMP 的数据库分页查询遵循的就是上面第三种方案，当需要进行分页查询或翻页时，页面端会将需要查询的页数、每页的记录数甚至总的记录数提交给服务器端，服务器端根据这些信息查询数据库，返回时再将这些更新过后的信息返回到页面端刷新分页的显示信息。

与 EMP 分页处理对象相关的服务器端的数据模型如下表所示：

```
用于分页处理对象显示时所需要的数据模型：
<kColl id="pageInfo">
    <field id="currentPage" />
    <field id="maxLine" />
    <field id="recordSize" />
</kColl>

翻页时，分页处理对象所提交的数据模型：

<kColl id="pageInfo">
    <field id="targetPage" />
    <field id="maxLine" />
    <field id="recordSize" />
```

```
</kColl>
```

pageInfo 表示的是一个 KeyedCollection，其中可能拥有的四个数据域分别是：

- currentPage：当前显示的页号(只用于显示分页信息)
- targetPage：需要转到或查询的页号(只用于服务器端查询。通常由页面端提交该值，服务器端根据该值进行查询)
- maxLine：每页的记录数。通过该值与 targetPage 就可以知道需要返回的记录是哪几条。比如 targetPage 为 2、maxLine 为 5，那么返回的记录就是第 6 条到第 10 条
- recordSize：总记录数。该值比较特殊，从理论上讲，为了能够尽可能的反应数据库中的数据状态，每页查询时都应该查询符合条件的总记录数。但是出于对性能的考虑，每次翻页时如果都要去查询总记录数会导致性能的急剧下降，而采取分页显示的一个需求就是为了提高性能。另外在实际应用中，应该是通过业务层次的设计来避免在查询过后查询记录发生改变的情况(或者改变的情况能够不影响其他用户的操作)，因为即使每次翻页都查询总记录数也无法避免已经查询出来的当前页的数据不会在用户进行相关操作之前发生变化。

  正是出于上述的考虑，EMP 的分页查询会分析该值，如果在数据模型中存在着该值，则不再查询总记录数，否则就会查询符合条件的总记录数。

2.3.4.1. 基本属性

- table：与分页处理对象相关联的列表数据标签操作对象
- previous：“上一页”BUTTON 按钮对象
- next：“下一页”BUTTON 按钮对象
- first：“首页”BUTTON 按钮对象
- last：“尾页”BUTTON 按钮对象
- jumpButton：“跳转”BUTTON 按钮对象
- jumpInput：设置跳转的指定页号的 INPUT 输入框对象
- maxLine：设置每页记录数的 INPUT 输入框对象
- currentPage：用于显示当前页的 SPAN 标签对象
- totalPage：用于显示总页数的 SPAN 标签对象
- recordSize：用于显示总记录数的 SPAN 标签对象
- info：当前分页处理对象所显示的分页信息，其结构如下：“{ currentPage：1，maxLine：15，totalPage : 1 , recordSize: 0 }”

2.3.4.2. 基本方法

- setInfo：更改分页处理对象所包含、代表的分页信息，其参数为：currentPage,maxLine,recordSize
- refreshInfo：根据分页处理对象中所包含的分页信息刷新页面上的分页显示
- toPage：转到指定的显示页，其参数为指定的页号
- toPrevious：转到上一页
- toNext：转到下一页
- toFirst：转到首页
- toLast：转到尾页
- directTo：转到 jumpInput 属性所对应的跳转框中指定的页面
- setWaiting：设置分页的各个操作元素处于查询等待状态(即不可操作状态)

2.3.5. EMP列表数据标签

EMP 列表数据标签就是与列表数据标签操作对象相对应的一个 EMP 标签，其实现类是 com.ecc.emp.ext.tag.EMPExtDataTable。另外，在 EMP 列表数据标签中所能包含的只能是 EMP 单数据标签。

2.3.5.1. 标签属性

- icollName：标签所要展现的集合数据(IndexedCollection)的名称，同时该名称也是标签的性标识
- cssClass：整个 TABLE 标签的样式，缺省是 emp_table
- cssTitleClass：标题栏所在 TR 的样式，缺省是 emp_table_title
- cssStatisticClass：统计栏的样式，缺省是 emp_table_statistic
- selectType：列表的选择方式：0-不能选择；1-单选；2-多选
- needTableTitle：是否需要显示标题栏，缺省是需要显示的
- editable：列表是否可编辑，缺省是不可编辑的。如果列表可编辑也不表示所有的字段都必须是可编辑的，可以通过设置列表中指定字段的 flat 属性为 false 使得特定的字段不可编辑
- statisticType：列表统计的类型。从理论上讲，每个字段都可以拥有不同的统计类型(如一个字段统计平均值，另一个字段统计总数，甚至于某个字段即统计平均值也统计总数)。但是从实际项目开发的分析得知，绝大部分的情况是一张表只会统计某种类型的统计值，多个的统计值反而导致了列表显示的混乱，因此在设计时决定每个列表只统计一种类型
- pageMode：列表是否分页，缺省是进行分页的。该属性应该与服务器端的名称为 pageInfo 的 KeyedCollection 对象配合使用(具体见分页处理对象章节介绍)

- url：分页查询的 url。显示列表页面的 url 与分页查询(或者称为刷新列表)的 url 是不一致的，显示列表页面的返回结果是一个 JSP 页面，在 JSP 页面中除了列表的数据信息之外还有列表的显示信息以及除列表之外的其它 JSP 显示信息。而分页查询的 url 则只返回列表的数据信息(还包括分页的信息)，其组织结构是一个 JSON 字符串。因此对于查询列表而言，通常会定义两个 MVC，一个用于显示列表页面，另一个用于分页查询(包括了针对指定列表的查询、翻页、刷新等)
- reqParams：分页查询的 url 中需要携带的参数

2.3.5.2. 配置示例

```
<emp:table icollName="testTable">
    <emp:text id="field1" label="编号" />
    <emp:text id="field2" label="金额" dataType="Currency"/>
    <emp:multilink id="multiOpLink" label="操作">
        <emp:link id="update" label="修改" opName="修改" operation="update"/>
        <emp:link id="delete" label="删除" opName="删除" operation="delete"/>
    </emp:multilink>
</emp:table>
```

2.4. 分组数据标签操作对象

分组数据标签操作对象与上面所提到的单数据标签操作对象以及列表数据标签操作对象存在着一个本质上的区别：分组数据标签操作对象并不对应于服务器端数据模型中的特定数据。分组数据标签操作对象的作用是为了将一个页面中的多个数据标签操作对象进行分组以便于管理。

例如多个的单数据标签操作对象可以分在一个分组数据标签操作对象中，这样就可以通过对分组数据标签操作对象来同时控制这些单数据标签操作对象。分组数据标签操作对象最大的应用场景是：提交时的校验 checkAll、复制 toForm 以及重置时的 reset。

分组数据标签操作对象的设置比较简单，对于整个页面而言，存在着一个最大的分组数据标签操作对象：_default。如果想要将某个区域内的所有数据标签操作对象(包括单数据标签操作对象和列表数据标签操作对象)放到一个分组对象中，那么可以在这些标签的最外围定义一个 DIV，该 DIV 惟一特殊的地方在于 class 属性为：emp_group_div。那么就会根据该 DIV 生成一个分组数据标签操作对象，并使用 DIV 的 ID 作为其惟一标识，在该 DIV 中的所有数据标签操作对象都会包含在分组数据标签操作对象中。

分组数据标签操作对象的访问方式是：“page.dataGroups.”+分组的名称，例如对于整

个页面而言，存在着缺省的最大的分组就是：page.dataGroups._default。

这里需要注意的是：到目前为止，并没有介绍如何访问单数据标签操作对象和列表数据标签操作对象，那是因为那两个对象并不能直接在 JS 进行访问，需要通过数据元素对象进行间接的访问到。而分组数据标签操作对象是由页面上人为的分组产生的，并不对应于一个特定的数据元素对象，因此为了能够比较方便的访问到，将该对象设置到 page.dataGroup 中。

2.4.1. 基本原理

分组数据标签操作对象的创建原理就是在创建其它数据标签操作对象时，判断在数据标签操作对象所对应的 HTML 元素的外围是否存在着特定的 DIV，其 class 属性为 emp_group_div(一直从当前标签判断到整个页面的最外围部分，通过 html_obj.parentNode 进行访问)。如果存在，则创建出一个分组数据标签操作对象，将当前对象添加到其中并继续向外围判断。因此一个标签操作对象可以属于多个的分组数据标签操作对象中。

2.4.2. 对象的基本属性和方法

2.4.2.1. 基本属性

- tag：分组数据标签操作对象所对应的 DIV 标签对象
- id：分组数据标签操作对象的惟一标识。由所对应的 DIV 标签对象的 ID 属性决定

2.4.2.2. 基本方法

- push：为分组数据标签操作对象添加一个标签操作对象
- _checkLength：批量的检查分组内的所有标签操作对象的输入数据的必输和字符长度
- _checkDataType：批量的检查分组内的所有标签操作对象的输入数据是否通过 dataType 校验
- checkAll：批量的检查分组内的所有标签操作对象的输入合法性
- toForm：批量的将分组内的所有标签操作对象复制到指定的表单中
- reset：批量的将分组内的所有标签操作对象进行重置

2.4.3. 配置示例

```
<div id="dataGroup_Test" class="emp_group_div">
    <table>
        <tr><td><emp:text id="Test.province" label="省份" required="true" onlyControlElement="true"/>请输入省份</td></tr>
        <tr><td><emp:text id="Test.city" label="城市" required="true" onlyControlElement="true"/>请输入城市</td></tr>
        <tr><td><emp:text id="Test.area" label="区县" required="false" onlyControlElement="true"/>请输入区县</td></tr>
    </table>
</div>
```

2.5. 标签操作对象的访问

标签操作对象的访问分为两种：一种是单数据标签操作对象和列表数据标签操作对象；另一种是分组数据标签操作对象。

单数据标签操作对象和列表数据标签操作对象的访问方式是通过 Field 数据元素对象和 IColl 数据元素对象来访问的。其示例如下：

```
var fieldObj = …;//如何获得 Field 数据元素对象请详见“Field 数据元素对象”章节
var obj = fieldObj._obj;//通过_obj 属性获得相关联的数据标签操作对象
//然后再根据得到的数据标签操作对象对页面的展现进行控制，如：
obj._renderHidden(false);//隐藏
```

分组数据标签操作对象的访问是直接通过 page 对象进行的，其示例如下：

```
var obj = page.dataGroups.objectName;//objectName 就是具体的分组数据标签操作对象的名称
//然后再根据得到的数据标签操作对象对页面的展现进行控制，如：
obj.checkAll();//对分组内所有数据标签操作对象进行校验
```

3. 数据元素对象

3.1. 概述

在“标签对象概述”章节中曾经提到了数据元素对象，该对象是参照于 EMP 服务器端的数据模型的结构进行设计的。EMP 的数据模型主要包括三种：IndexedCollection、KeyedCollection、DataField，相应的在页面端也设计了三种数据元素对象：IColl、KColl、Field。

3.1.1. 目的

数据元素对象设计的初衷是为了使得开发人员更容易理解及操作在页面端上“注册”的各类对象。对于使用 EMP 平台进行开发的开发人员来讲，其数据模型是特色非常鲜明的一个结构，而通过将页面上“注册”的各类对象也使用服务器端的数据模型方式进行组织可以降低开发人员的学习成本。另外一个原因是通常开发人员在设计服务器端的数据模型时就已经包含了一定的业务含义，那么通过对页面端的数据元素对象的封装，也能使得页面上的数据包含着一定的业务含义，便于管理、控制，而不仅仅是把页面上的标签进行零散的显示。

简单的说，最主要的目的是为了让页面端的数据也能够像服务器端的数据模型一定被组织、管理和操作。

3.1.2. 实现原理

数据元素对象的实现原理是依据服务器端对于数据存取的规范进行创建的。在服务器端，如果一个存取一个数据时，数据名称有着相应的规范：

- 数据名称的格式是：aColl[1].bField，则该数据模型的结构是：

```
<iColl id=”aColl”>
    <kColl>
        … …
    </kColl>
    <kColl>
        <field id=”bField” />
        … …
    </kColl>
```

```
    … …
</iColl>
```

其中的 aColl 表示的是一个名称为 aColl 的 IndexedCollection 对象，而“[]”中括号中的数值则代表着集合中第几条记录，剩下的“.”符号后面的部分则表示字段名

- 数据名称的格式是：aColl.bField，则该数据模型的结构是：

```
<kColl id=”aColl”>
    <field id=”bField”/>
</kColl>
```

与上面一种格式相比，这种数据名称的格式少了“[]”符号，其中“.”符号前面的部分表示的是一个 KeyedCollection，符号后面的部分则表示为数据域名称

- 除了上面两种之外的名称，则表示为简单的 DataField 对象

与上述数据名称的格式的机制一样，在解析标签操作对象时，如果某一个标签操作对象的名称为 aColl.bField，那么会根据上述的格式创建出一个名称为 aColl 的 KColl 数据元素对象和一个名称为 bField 的 Field 数据元素对象。当需要访问具体的 bField 对象时采用的方式就是直接的“aColl.bField”的方法。除此之外，对于 IndexedCollection，在页面端并不是真正的生成“aColl[0].bField”格式的标签操作对象，而是根据列表数据标签操作对象来创建 IColl 数据元素对象。

由上面可见，单数据标签操作对象与 Field 数据元素对象、列表数据标签操作对象与 IColl 数据元素对象之间有着直接的关联关系，可以通过各自的引用访问对方：**在数据元素对象中可以通过“_obj”访问标签操作对象；而在标签操作对象中可以通过“data”属性访问数据元素对象**。

比较特殊的是 KColl 数据元素对象，因为没有相应的数据标签操作对象，所以也就没有相应的关联关系。

3.1.3. 访问方式

在设计之初，为了方便访问数据元素对象，便将数据元素对象按照其数据模型的结构设置成父对象的一个属性，对于最外层的结构，则将 window 对象作为父对象。例如一个名称为 aColl.bField 的标签操作对象，则会生成一个 KColl 数据元素对象并以 aColl 为名称作为 window 对象的一个属性，同时生成一个 Field 数据元素对象并以 bField 为名称作为 aColl 对象的一个属性。因此就可以通过 aColl 访问到 KColl 数据元素对象，也可以通过 aColl.bField 访问到 Field 数据元素对象。

也正是上面这种特殊的访问方式，导致了数据元素对象中的属性名称及方法非常的少，而且要求相应的数据模型的名称与数据元素对象中的属性名称、方法不能起冲突(即不能同名)。从原则上讲，数据元素对象应该不包含其他的属性和方法，但是对于某些常用的方法，

又不希望通过关联关系再进行操作，因此在数据元素对象中又封装了个别的属性和方法，这些属性和方法都是以“_”开头的。

3.2. Field数据元素对象

Field 数据元素对象与单数据标签操作对象之间存在着直接的关联关系，单数据标签操作对象都必须通过 Field 数据元素对象才能访问到。

3.2.1. 保留属性和方法的名称

- _obj：与 Field 数据元素对象相关联的单数据标签操作对象
- _setValue：对相关联的单数据标签操作对象进行设值
- _getValue：获取相关联的单数据标签操作对象的真实值
- _toForm：将相关联的单数据标签操作对象的取值复制到指定表单 form 中
- _checkAll：对相关联的单数据标签操作对象进行校验
- _isField：判断当前对象是否是 Field 数据元素对象

3.3. IColl数据元素对象

IColl 数据元素对象与列表数据标签操作对象之间存在着直接的关联关系，列表数据标签操作对象都必须通过 IColl 数据元素对象才能访问到。

3.3.1. 保留属性和方法的名称

- _obj：与 IColl 数据元素对象相关联的列表数据标签操作对象
- push：往 IColl 数据元素对象中添加 KColl 数据元素对象。虽然没有强制的限制 push 的参数是 KColl 数据元素对象，但从页面框架的设计上和服务器端的数据模型上考虑，IColl 数据元素对象中只能包含 KColl 数据元素对象
- splice：删除 IColl 数据元素对象中指定的记录号的 KColl 数据元素对象。其中只有一个参数 idx 就是所指定的记录号。该方法与 Array 的 splice 略有不同，没有实现 splice 的第二个参数。
- _getSize：获得当前 IColl 数据元素对象中所拥有的总记录数(KColl 数据元素对象的总数)
- _toForm：将 IColl 数据元素对象中所包含的所有记录复制到指定的表单 form 中(具体 KColl 数据元素对象如何复制到指定表单请详见“KColl 数据元素对象”章节)
- _checkAll：校验 IColl 数据元素对象中所包含的所有记录(具体 KColl 数据元素对象如何校验请详

见“KColl 数据元素对象”章节)

- _addKColl：添加一种记录到 IColl 数据元素对象中，并同时将结果展现到页面上(即可以在页面上看到列表中添加了一行)。其中的参数是 KColl 数据元素对象。
- _isIColl：判断当前对象是否是 IColl 数据元素对象

3.3.2. 对象结构

为了与服务器端访问 IndexedCollection 对象保持一致，在页面端访问 IColl 数据元素对象内部的 KColl 数据元素对象时所采用的方式就是：“bKColl = aIColl[0];”。从其中可以看到，在 IColl 内部，KColl 是作为 IColl 的一个属性存在，且属性名为 0、1、2---n。具体的可见下表：

IColl 数据元素对象	
_obj	列表数据标签操作对象
其它保留属性名称	
0	KColl 数据元素对象
…	
size-1	KColl 数据元素对象

3.4. KColl数据元素对象

KColl 数据元素对象与上述两种数据元素对象有着最大的不同之处：上述两种数据元素对象都会对应着一种数据标签操作对象，而 KColl 数据元素对象而没有相关联的数据标签操作对象。KColl 数据元素对象与服务器端的 KeyedCollection 一样，只是起到封装一条记录或者是分类一组数据的作用。另外，KColl 数据元素对象中可以包含任何的数据元素对象，包含 IColl 数据元素对象、KColl 数据元素对象和 Field 数据元素对象。

KColl 数据元素对象的分组作用与分组数据标签操作对象的分组作用不一样：分组数据标签操作对象的分组机制是将页面中处在同一块区域内的所有数据标签操作对象进行分组；而 KColl 数据元素对象机制是通过单数据标签操作对象中拥有相同的分组名称(即符号“.”之前的名称相同)或处于 IColl 数据元素对象的同一记录内。

3.4.1. 保留属性和方法的名称

- _checkAll：校验 KColl 数据元素对象中所包含的所有数据元素对象(包括 IColl、KColl、Field 数据对象)

- _toForm：将 KColl 数据元素对象中所包含的所有数据元素对象复制到指定的表单 form 中(包括 IColl、KColl、Field 数据对象)
- _putKColl：对 KColl 数据元素对象中同名的 Field 数据元素对象进行更新(更新的只包含其中的 Field 数据元素对象)。该方法的参数是一个 KColl 数据元素对象
- _isKColl：判断当前对象是否是 KColl 数据元素对象

3.4.2. 对象结构

与服务器端的 KeyedCollection 结构相同，在 KColl 数据元素对象中可以包含 IColl 数据元素、KColl 数据元素和 Field 数据元素。这些数据元素是作为一个属性在 KColl 数据元素对象中存在，其中的属性名称就是数据元素的名称。具体的结构如下表如示：

KColl 数据元素对象

保留属性名称

the name ---------- 数据元素对象

…

the name ---------- 数据元素对象

需要注意的是：从上表中能够发现，**KColl 数据元素对象中所包含的数据元素对象的名称不能与 KColl 的保留属性名称相同**。例如：在 KColl 数据元素对象中有一个保留属性名称_isKColl，那么在该 KColl 数据元素对象中就不能包含一个同名的数据元素对象，否则在调用“aColl._isKColl”就无法确认返回的对象是什么了(更确切的说如果有同名的数据元素对象，那么原先的保留属性就会被覆盖，导致 KColl 数据元素对象结构被破坏而无法被其它对象确认)

3.5. 数据元素对象的访问

数据元素对象在创建时都会“注册”到 window 中，作为 window 对象的一个属性存在，因此在访问数据元素对象时可以直接使用数据元素对象的名称。示例如下：

```
//对于<emp:text id="bField" />而言，访问方式如下：
var value = bFfield._getValue();

//对于<emp:text id="aColl.bField" />而言，访问方式如下：
var value = aColl.bField._getValue();
```

```
//对于<emp:table  iCollName=”aColl”><emp:text  id=”bField”/></emp:table>而言，访问第一条记录中的
//bField 的方式如下：

var value = aColl[0].bField._getValue();
```

4. 与数据无关的其它组件

4.1. 布局组件

对于 HTML 编辑而言，我们可以通过 TABLE、DIV、UL、LI 等等标签以及 JS、CSS 等定义实现非常复杂、用户体验非常丰富的 UI 展现，甚至于针对不同的用户还能自定义拥有修改的展现。但是，对于大部分的页面或者说在用户体验非常丰富的 UI 展现中的局部区域，页面的展现是拥有一定的规律的。最普遍的情况就是将数据按照多行一列或多行两列的方式进行排列，然后再将这些有规则的展现通过其它的配置统一在一个页面上进行展现就能达到相应的 UI 展现。

在这里提高的布局组件就是为了能够更加方便的开发这种有规则的页面布局，通过简单的配置，能够达到页面上展现单列、两列乃至多列的数据展现。同样的，对于无规则的页面展现，这里的布局组件就无法胜任了，但是开发人员可以通过对布局组件的进一步排列、设置实现更复杂的用户体验。

4.1.1. 基本原理

布局组件就是根据定义在其中的 EMP 数据标签的相关配置，自动的构造出一个 TABLE 表格结构。根据每个标签的 colSpan 属性决定所占的列数以及根据 CSS 属性等决定显示的样式等等。

其中如果某一列隐藏，则同一行的其它列则自动的扩展；如果某一行中所有列都隐藏，那么则该行就自动隐藏。这是为了保持页面整体展现不是随着个别标签的显示、隐藏而出现变动，某一列隐藏后，只影响到当前行的显示。

需要注意的是：**布局组件只能对 EMP 单数据标签进行组织、布局，对于其它的标识无法标识**。

4.1.2. 属性介绍

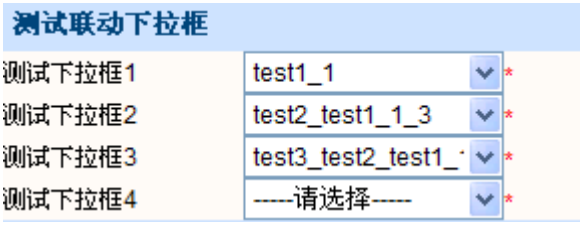
- id：布局组件的惟一标识。并且为每个布局组件注册一个分组标签操作对象，以便于统一的管理同一布局组件中的标签操作对象
- maxColumn：表格布局的最大列数，缺省是两列显示的。这里说的列数并不是指 TABLE 中的 TD 标签，则是将描述、数据展现和说明等与一个数据标签相关联的部分当作一列
- title：布局组件的标题显示，如果不定义该属性，则不显示任何标题

- cssClass：布局组件中整个表格的样式，缺省是 emp_gridLayout_table
- cssTitleClass：布局组件中标题的样式，缺省是 emp_gridlayout_title

4.1.3. 配置示例

```
<emp:gridLayout id="TestGroup" maxColumn="1" title="测试联动下拉框">
    <emp:select id="User.test1" label="测试下拉框 1" required="true" defvalue="test1_1"/>
    <emp:select id="User.test2" label="测试下拉框 2" required="true" defvalue="test2_test1_1_3"/>
    <emp:select id="User.test3" label="测试下拉框 3" required="true" />
    <emp:select id="User.test4" label="测试下拉框 4" required="true"/>
</emp:gridLayout>
```

上表的配置示例在页面中的展现如下图所示：



4.2. 数据类型组件

在实际的项目开发过程中，有两个问题是基本上都会面临到的：如何对页面上的数据进行校验以及如何对真实值进行修饰转成用户体验比较好的显示值？这两个问题在 EMP 平台中都统称为数据类型。

对于大部分的数据而言是没有所谓的数据类型的概念的，其无论是显示还是提交都是一段字符串，例如像用户名、地址等。但是对于某些数据，却有着相应比较严格的格式，例如：金额、电子邮箱地址、身份证号码等等，对于这类数据并不是用户随便输入的数据就一定是正确的，那么这就存在着数据校验的问题。同时像金额，通常的显示方式都会是整数部分每三位用“,”号进行分割而不是单纯的显示一个数字，那么这些数据就存在着真实值与显示值之间的转换。

对于数据类型的实现，EMP 平台将其作为平台的一个重要组成部分进行管理，而不是要由各个开发人员自行开发。数据类型组件又分为页面端的数据类型实现和服务器端的数据类型实现，其中页面端的数据类型实现主要通过 JS 代码来完成，而服务器端的数据类型则是通过对指定数据类型接口的实现来完成的。

4.2.1. 页面端的实现

对于页面端的数据类型实现比较简单，不需要继承任何的实现类，只需要保证在数据类型对象中包含两个方法：check 和 display。

其中 check 方法表示的是数据类型的校验功能，其中至少有三个参数：一个参数是输入值，即用于校验的值；另两个参数则是校验失败的信息(formatMsg 和 rangeMsg，这两个信息将在服务器端的实现中介绍)。**需要注意的是 check 方法返回值是一个特定的对象：EMPTools.message**(该对象的具体属性见 JS 工具类中的介绍)。在 check 方法中不仅仅是对输入值进行校验，如果校验成功还需要将真实值当作 EMPTools.message 中的 message 属性返回

display 方法表示的则是数据类型的转化。这里说的转化指的是从真实值转化为显示值(显示值转化为真实值由 check 方法完成)。该方法至少包含一个参数：真实值，同时该方法的返回值为相应的显示值。这里需要注意的是，**调用 display 的真实值应该是已经确认为有效的真实值，即在 display 方法中不再对真实值进行校验**。

4.2.2. 服务器端的实现

服务器端的数据类型实现需要继承 com.ecc.emp.datatype.EMPDataType 抽象类。在该抽象类中提供了五个抽象方法必须进行相应的实现，下面进行一一的介绍：

- getJavaTypeName：该类型的数据对应于 JAVA 类型的名称，如数字类型的数据可能就会对应 JAVA 中的 Integer、Double 等类型(该方法主要用于 WebService 渠道接入中)
- validateStringValue：对输入的数据进行校验。在抽象类中有两个同名的校验方法，一个带着参数 Locale 表示的是根据本地化相关信息进行校验；另一个不带着参数则表示按照缺省的方式进行校验(通常的做法是先获得缺省的 Locale 对象，然后再调用需要 Locale 的校验方法)。在这里需要注意的是：**对于要校验的输入值可能是真实值，也可能是显示值**，因此在校验时要同时考虑这两种可能性。如果校验不通过，则抛出一个 com.ecc.emp.datatype.InvalidDataException 异常，校验通过则返回 true
- convertFromString：对输入的数据进行转化，这里指的转化是指将输入值转化成真实值，其中的输入值是未经过校验的真实值或显示值。因为输入值是未经过校验的，因此通常都会在该方法开始的地方先调用一下 validateStringValue 方法进行一下校验。除了需要校验之外，还需要注意的一点是：**每个数据类型都有一个 keepStringValue 属性，如果该属性为 true，则返回的真实值是一个字符串；而若是 false，则返回的真实值应该是一个相应的 JAVA 对象**(与 getJavaTypeName 相对应)
- getStringValue：将真实值转化为显示值，在这里已经默认为真实值是正确的，无须进行校验。与 convertFromString 一样，根据 keepStringValue 属性的不同，真实值的对象可能是一个字符串，也可以是相应的 JAVA 对象，因此在转化之前需要先判断其参数中的真实值对象的类型。

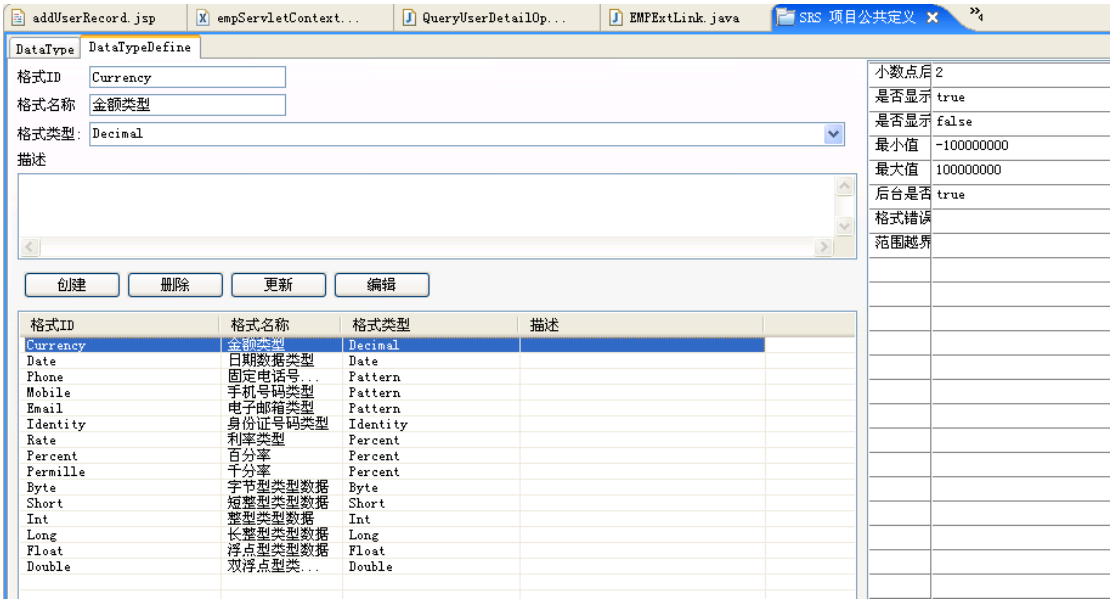
除了上面介绍的 keepStringValue 属性之外，每个数据类型都有另外两个属性：**formatErrorMsg 和 rangeErrorMsg**。这两个属性分别用于校验失败时的失败信息显示内

容。其中 formatErrorMsg 表示输入的格式错误，而 rangeErrorMsg 则表示输入的数据超出了指定范围。在实际应用中，这两个属性的值可以使用多语言中的关键字来表述，如果不定义这两个属性，则通常情况下页面端会去查找在多语言中有没有以这两个属性名称为关键字的多语言定义，如果有则使用多语言的描述，如果没有则由具体的页面端标签实现进行处理。

通过对上面介绍的几个抽象方法的实现，就能保证对输入值的校验以及真实值与显示值之间的相互转化。这里只是介绍了基本的实现方式，如果需要扩展出一种新的数据类型则详见“数据类型扩展”章节。

4.2.3. 配置示例

这里介绍的配置示例主要是 IDE 开发工具的使用为主，再辅以所生成的配置文件的基本内容来说明相应的配置。



从上图可以看到，当需要配置一个数据类型的具体定义时，需要知道该定义是基于哪种数据类型的，如金额就是基于 Decimal 数据类型的，利率、百分率和千分率就都是基于 Percent 数据类型的。创建了具体的定义之后，再根据数据类型中所拥有的属性进行配置，如最大值、最小值等等。

对于这些配置，在相应的 dataTypeDef.xml 文件中的示例如下：

```
<Decimal id="Currency" name="金额类型" showSymbol="false" keepStringValue="true" scale="2" showComma="true" max="100000000" min="-100000000"/>
```

其中 XML 标签的名称就指定了该定义所属的数据类型名称，其余的属性则是表明该定义所拥有的相应配置。

具体平台中拥有哪些数据类型名称，则可以从 dataType.xml 中获取：

```
<dataType id="Byte" name="Byte" implClass="com.ecc.emp.datatype.ByteType" >
    <document>字节型类型数据</document>
    <attributes>
        <attr id="max" name="最大值" defaultValue="127" editable="true"/>
        <attr id="min" name="最小值" defaultValue="-128" editable="true"/>
        <attr id="keepStringValue" name="后台是否转化为字符串形式" attrType="boolean" defaultValue="false" editable="true"/>
        <attr id="formatErrorMsg" name="格式错误时的提示信息" editable="true"/>
        <attr id="rangeErrorMsg" name="范围越界时的提示信息" editable="true"/>
    </attributes>
    <validateJS function="EMP.type.Number.check(displayValue, $max;, $min;, formatErrorMsg, rangeErrorMsg)" />
    <convertorJS function="EMP.type.Number.display(realValue)" />
</dataType>
```

在这里需要说明的是：数据类型代表着只是一种格式，并没有对这种格式中的具体属性进行限制，例如 Integer 数据类型只是代表着整数格式的数据。而对于每一种格式的数据而言，又可以定义出多个的拥有不同属性的数据类型定义。例如同样是整数格式的数据，数据类型定义的最大值就可以是不一样的，这样就可以将页面端或服务器端所使用的数据类型定义统一在一个地方进行配置，当项目环境发生改变之后，也可以通过统一的配置位置对所有的数据类型相关的信息进行配置。

4.3. 联动下拉框组件

把联动下拉框组件放到这个地方而不是“标签操作对象”章节，那是因为联动下拉框组件并不会生成真正的下拉框对象。它的作用只是用于标识页面中已经存在着某些下拉框存在着联动的效果以及联动的相关配置。也就是说，联动下拉框组件起作用的前提是，在页面中已经定义了需要进行联动的 emp:select 标签，否则联动下拉框组件就无法起作用了。

也正因为联动下拉框组件并不生成特定的 HTML 标签，因此在页面编辑中，联动下拉框组件的定义可以放在页面的任何位置(只要保证能够被包含在 emp:page 标签中—这是绝大多数的 emp 标签都必须满足的条件)。

4.3.1. 基本原理

当页面中定义了联动下拉框组件后，就会在 page.objectsDefine 对象中根据联动下拉框

组件的相关属性，生成 relatedSelects 属性，如下图所示：

```
if(!page){
	var page = new EMP.util.Page();
}
page.objectsDefine = {
	dataFields : [],
	dataTables : [],
	relatedTabs : [],
	relatedSelects :[{id:'linkSelectGroup',url:'updateUserSelect.do',needCache:'true',selects:['User.test1','User.test2','User.test3','User.test4']} ],
	dataDics : {}
};
page.renderEmpObjects();
```

在页面进行解析、注册各种标签操作对象结束后，会根据 page.objectsDefine 对象中的 relatedSelects 属性创建出 EMP.widget.RelatedSelectGroup，并且在页面上查找相应的 emp:select 标签所生成的单数据标签操作对象。如果有相应的标签操作对象，则将联动下拉框组件的对象“注册”到单数据标签操作对象中，这样当单数据标签操作对象所对应的下拉框取值发生改变之后，就会调用所“注册”的联动下拉框组件的对象进行处理，刷新与其联动的其它下拉框。

4.3.2. 标签属性

关于联动下拉框组件主要分为两个标签，其中一个是联动下拉框分组组件，另外一个是联动下拉框定义组件。

4.3.2.1. 联动下拉框分组标签

联动下拉框分组标签是确定一个联动下拉框分组的范围以及用于联动的一些属性

- id：惟一标识(与其它标签的 ID 需要区分)，对于联动下拉框组件对象(具体详见联动下拉框组件对象的介绍)的访问与数据元素对象一样，都是作为 window 中的一个属性存在，其中的属性名称就是 ID 的值
- url：下拉框的值改变时，需要发起请求的 url。当发起请求时，会将当前下拉框以及之前的联动下拉框的取值作为 url 的参数提交到服务器端
- reqParams：发起的请求中需要携带的参数。这个参数指的是固定的参数，即不随着下拉框内容的改变而改变。url 加上 reqParams，再加上当前下拉框及之前的联动下拉框中的取值就构成了完整的请求

- needCache：是否对已经获取的下拉框内容进行缓存。当下拉框取值发生改变时都会异步的请求服务器端的信息更新相应的联动下拉框。而在实际应用中，用户可能会不停的在几个选项之间选择，那么就会导致大量的异步请求。这个时候如果允许对已经存在的内容进行缓存，那么当选择一个之前已经选择过的选项时就直接使用缓存中的内容刷新联动下拉框，而不会重新发起请求。缺省情况是允许缓存的

4.3.2.2. 联动下拉框定义标签

联动下拉框分组标签主要作为是以联动下拉框定义标签进行分组，而联动下拉框定义标签则是用于定义当前页面中哪些下拉框能够存在着联动的效果，而且联动的顺序与联动下拉框定义标签在联动下拉框分组标签中的位置决定。联动下拉框定义标签的属性有：

- id：定义页面中存在的指定下拉框作为联动下拉框

4.3.3. 配置示例

```
<emp:relatedSelectGroup url="updateUserSelect.do" id="linkSelectGroup">
        <emp:relatedSelect id="User.test1"/>
        <emp:relatedSelect id="User.test2"/>
        <emp:relatedSelect id="User.test3"/>
        <emp:relatedSelect id="User.test4"/>
</emp:relatedSelectGroup>
```

上表中联动下拉框能够起作用的前提条件就是在页面中已经存在着四个 EMP 下拉框标签，同时联动的效果与联动下拉框定义标签的顺序有关，即 User.test1 的取值发起改变时，按照顺序的改变后面的三个下拉框的显示内容。

4.3.4. 联动下拉框组件对象

如果单纯只是为了展现联动下拉框的联动效果，那么前面的设计就已经足够了。但是为了对联动下拉框进行统一的管理以满足更加复杂多变的 UI 展现，专门设计了一个联动下拉框组件对象，该对象所对应的是 EMP 联动下拉框分组标签，即一个 EMP 联动下拉框分组标签会自动的创建出一个联动下拉框组件对象。

4.3.4.1. 基本属性

- id：联动下拉框组件对象对惟一标识

- url：联动下拉框组件对象刷新下拉框选项内容时异步调用的 url 请求
- selects：联动下拉框组件对象所包含的相关下拉框信息集合，其中每一个元素都是代表下拉框的 EMP 单数据标签操作对象
- firstSelect：联动下拉框组件对象中第一个下拉框的 EMP 单数据标签操作对象
- needCache：是否需要缓存下拉框的选项内容
- cache：缓存的内容。即使 needCache 为 false，该属性依然有值，只是其中的值不包括下拉框的相关选项内容

4.3.4.2. 基本方法

- registRelatedSelect：将一个下拉框对应的 EMP 单数据标签操作对象“注册”到联动下拉框组件对象中，其中参数是相应标签操作对象的名称
- doInitSelectContent：初始化指定下拉框的内容，其中的参数是需要进行初始化的下拉框所对应的 EMP 单数据标签操作对象
- updateSelectInnerHTML：更新下拉框的选项内容，其中第一个参数是需要进行更新的下拉框所对应的 EMP 单数据标签操作对象，第二个参数是用于更新的对象，第三个参数则是导致更新的前面联动下拉框的取值(该选项通常用于缓存的关键字)

通过上面的介绍也可以得知，开发人员如果不想使用联动下拉框组件的相关标签进行定义，那么也可以直接通过联动下拉框组件对象对页面中已经存在的下拉框进行“注册”，使之具有联动的效果，但是这种方式要求开发人员对联动下拉框有比较清晰的认识，因此不推荐使用

4.4. 页签组件

在具体介绍页签组件之前，先大概介绍一下什么是页签组件。页签组件简单的讲就是在一个页面中显示多个请求的结果页面，通过类似于选项卡的方式在多个页签之间进行跳转。具体的页面展现如下图所示：

上图表示的是一个 JSP 页面，在这个 JSP 页面中又包含着两个页签(或者称为两个选项卡)，每个页签都代表着一个请求的返回页面。当点击页签的标题(在上图中是：标签 1 和标签 2)时，JSP 页面就可以分别显示相应请求的返回页面。

4.4.1. 基本原理

页签组件在生成页面的 HTML 代码时将会生成一个大体的框架，例如上图中所示的有标题行，有内容显示的区域等。在生成这个大体框架之后，页签组件会在 page.objectsDefine 中定义 relatedTabs 属性，在这个属性中定义了当前存在的页签分组以及分组中各个页签选项的信息。最后在页面载入之后，对页面组件生成的框架进行处理，添加必要的事件和 CSS 样式，例如对页面的标题添加 click 事件，这样在点击标题时就能够对页签的显示区域进行替换。

页签的显示内容主要分为两种：一种是在整个 JSP 页面载入时页签中的内容就已经存在了，页签的作用只是为了对内容进行重整显示；另一种是页签中的内容还未载入，需要向服务器端重新发起请求。对于第一种情况，页签中的内容使用一个 DIV 来展现，而对于另外一种则使用一个 IFRAME 来实现。

页签组件生成的框架大体如下图所示：

```
<div id='relatedtabs_SLogTabs_tabs' mainTab='tab1'>
   <a>页签 1</a>
   <a>页签 2</a>
</div>
<div id='relatedtabs_SLogTabs_main'>
```

```
  <div id="tab1_div"    label='页签 1' initial='true' url='/SRS/signOn.do'>
      <iframe />
  </div>
  <div id="tab2_div"    label='页签 2' initial='false' url='/SRS/tab2.do'>
      <iframe />
  </div>
</div>
```

4.4.2. 标签属性

页签组件所涉及到的标签主要有两个：页签分组标签和页签定义标签(有点类似于联动下拉框组件)。

4.4.2.1. 页签分组标签

页签分组标签主要作用是对页面进行分类，一个页签分组标签就表示了在一个区域内对页面的展现。

- id：惟一标识(与其它标签的 ID 需要区分)，对于页签组件对象(具体详见页签组件对象的介绍)的访问与数据元素对象一样，都是作为 window 中的一个属性存在，其中的属性名称就是 ID 的值
- mainTab：页签分组中主页签的 ID，该属性通常都用于表示在页面载入时缺省显示哪个页签的内容，如果主页签的内容还未载入，那么就会自动发起请求载入主页签的内容

4.4.2.2. 页签定义标签

页签定义标签是对页签分组中每个页签的详细设置，例如标题的名称、页签内容的来源等等

- id：页签分组中每个页签的惟一标识
- label：页签所显示的标题
- url：页签所对应的请求 url
- initial：是否在 JSP 主页面载入时就对页签进行初始化(发起请求)，缺省是 false
- needFlush：每次点击标题时，是否都要刷新页签的内容(是否都要重新向服务器发起请求)，缺省是 false

在这里需要介绍的是 url 属性，如果没有定义 url 属性则表示当前页签的内容随着主页面载入时就已经载入了，不需要再向服务器端重新发起请求。而如果定义了 url，则在页签初始化时页签组件对象会在该页面中自动的构造出一个 IFRAME 用于载入页签的显示内容。

4.4.3. 配置示例

```
<emp:tabGroup mainTab="tab1" id="tabGroup">
    <emp:tab label="IFRAME 方式页签" id="tab1" url="welcomeInfo.do" needFlush="true" initial="true">
    </emp:tab>
    <emp:tab label="DIV 方式页签" id="tab2">
        <!-- DIV 中具体的内容 -->
    </emp:tab>
</emp:tabGroup>
```

在上表中，分别定义了两种不同类型的页签，一种是在 JSP 主页面载入时就已经存在着页签内容，另外一种是在需要显示页签内容时才会动态的载入页签的内容，而且根据 needFlush 属性，当每次点击标题时都会重新载入页签的内容

4.4.4. 页签组件对象

与页签组件拥有两种标签一样，页签组件也拥有两个对象：页签分组对象和页签定义对象。

将页签组件“注册”成页签组件对象的主要作用是将页签放到整个 EMP 前端组件体系中进行管理，通过对页签组件各个对象的操作，可以实现对页面的复杂控制，例如可以动态的在页面端控制页签的显示与否等等。

4.4.4.1. 页签分组对象

- id：页签分组对象的惟一标识，可以在 window 对象中直接通过该 ID 访问到页签分组对象(与数据元素对象一样)
- tabs：页签分组对象中所包含的页签定义对象集合，可以通过 tabs[“your tabName”]或者是 tabs.+your tabName 来访问具体的页签定义对象
- tabsDiv：包含页签分组中所有页签标题的 DIV 标签对象。该对象与 mainDiv 构成了页签分组的全部区域
- mainDiv：包含页签分组中所有页签显示内容的 DIV 标签对象
- mainTab：页签分组中主页签的名称
- loading：在页签完全载入之前，用于显示“请等待”信息的 DIV 标签对象，该对象对 firefox 等

浏览器不起作用

- _parseParams：对页签组件标签所生成的 HTML 结构进行解析
- _addTab：在页签分组中增加一个新的页签，该方法有两个参数：第一个是新页签的 ID 标识，第二个参数是新页签的属性对象(以{}方式定义的对象)。其中第二个参数只在动态添加一个新的页签时使用

4.4.4.2. 页签定义对象

- id：页签对象的惟一标识
- label：页签对象的描述，该属性的值能够显示在页签标题栏中
- relatedTabGroup：页签对象所在的页签分组对象
- needFlush：是否每次点击页签标题时都刷新页签的内容
- initial：是否在载入主页面时初始化页签的内容
- loaded：当前页签内容是否已经载入完成
- _clickLink：点击页签时所触发的 JS 方法
- refresh：刷新或生成页签的内容
- _readyStateChange：页签内容载入完成后调用的 JS 方法，只针对 IE 有效
- renderHidden：处理页签的隐藏与否，其中参数为 true/false，true 表示隐藏

4.5. 操作按钮组件

操作按钮组件是对普通按钮的一个简单的封装，其主要的功能是提供了一种可扩展的前端按钮的权限控制。可扩展在于操作按钮组件只提供了用于权限控制的接口而没有限制必须采用的权限内容，而前端的意思是其权限是在前端页面通过 JS 的方式进行控制。

4.5.1. 基本原理

其原理也相对比较简单，对于需要进行权限控制的按钮，在生成页面时，其按钮缺省为隐藏。然后在页面载入后对页面内所有的按钮一一进行权限判断，如果是有权限，则将按钮

显示出来。

其中对按钮进行权限判断的方式是在最外层(或最底层)的页面中定义名称为 checkPermission 的 JS 方法。该 JS 方法有两个参数，第一个参数 menuId 表示当前菜单或模块的 ID 标识，第二个参数 op 表示按钮所表示的操作 ID。该方法需要返回 true\false，如果返回 true 就表示指定菜单下的指定操作拥有相应的权限

例如：

```
function checkPermission(menuid, opid) {
        var item = operations[menuid];
        if (item){
            if(item[opid]){
                return true;
            }
        }
        return false;
};
```

另外，每个按钮都缺省会触发一个 JS 方法，通过编辑该 JS 方法可以达到按钮所要完成的相应功能

4.5.2. 属性介绍

- id：当前按钮的惟一标识，并且指定了该按钮所触发的 JS 方法："do+id(其中 id 中首字母大写)"为名称的 JS 方法。如 id 为 update，则触发 doUpdate 方法
- label：按钮上所显示的描述字符串
- op：按钮所对应的操作权限名称(如果该属性为空，则表示不需要进行权限的控制)

4.5.3. 配置示例

```
<emp:button id="viewSLog" label="查看" op="view"/>
```

4.5.4. 特殊的emp:returnButton标签

在操作按钮组件中有一个比较特殊的扩展按钮：emp:returnButton，该按钮只适用于典型表模型的弹出页面中，用于将弹出页面中的值返回到主页面。

该标签会触发指定的 doReturnMethod 方法，该方法的参数有两个：

- popReturnMethod：需要调用的主页面中的方法的名称，该值是由弹出窗口的 url 中的 popReturnMethod 参数决定
- element：按钮本身的对象

该标签使用时主要是要对 doReturnMethod 方法进行编辑，通常情况下，在方法中需要做到几个事情：

- 处理当前页面中的业务操作
- 获得需要返回主页面的数据(通常以{id:value;id:value}这种值对的方式返回)
- 调用主页面中名称为 popReturnMethod 的方法
- 关闭当前窗口

```
function doReturnMethod(methodName){
    if (methodName) {
        //处理当前窗口中的业务操作
        //在本例子中，当前窗口无任何业务操作

        //获得返回主页面的数据
        var data = {column11:'33'};
        //调用主页面中的方法
        var parentWin = EMPTools.getWindowOpener();
        eval("parentWin."+methodName+"(data)");
        //关闭当前窗口
        window.close();
    }else{
        alert("未定义返回的函数，请检查弹出按钮的设置!");
    }
};
```

4.6. emp:form表单标签

表单标签的主要作用是为了简化表单的开发(减少 JS 代码的编写)，以及提供一些必要的表单元素(如 sessionId)等，另外也是为了更符合一般的页面编码习惯

4.6.1. 基本原理

与普通的 form 不一样的地方在于，**emp:form 表单标签只能提交被 emp:form 标签所包含的 emp 标签定义的数据(包括列表数据)**，而对于其它数据(如使用普通的 input 输入框)需要采用 JS 在提交的表单中生成一个同名的隐藏域。

使用 emp:form 标签会在标签结束的时候生成一个隐藏的 HTML 表单 form 表单，在该表单中包含了一些必要的表单元素，如 sessionId 等。同时会生成一个 DIV 标签来代替 emp:form 标签所在的位置，而该 DIV 标签 id 属性的值为：“dataGroup_in_form”+表单 ID、class 属性的值为：“emp_group_div”，根据前面介绍的“分组数据标签操作对象”可知，该 DIV 标签会在页面端“注册”一个分组数据标签操作对象，并且将其中所包含的所有 emp 标签所生成的标签操作对象。

除此之外，还会生成两个缺省的 doSubmit 和 doReset 方法。在这两个方法中，将 emp:form 标签所生成的“分组数据标签操作对象”进行 checkAll、toForm 和 reset 处理(即将分组数据标签操作对象中所包含的所有 emp 标签所代表的数据 toForm 到隐藏的 form 表单中，而其它想要提交的数据也需要使用 JS 创建出一个隐藏域的方式 append 到隐藏的 form 表单标签中)。

如果页面提交或重置时，需要使用所生成的缺省 doSubmit 和 doReset 方法(即在页面中不需要进行 JS 编程时)，可以将相应的操作按钮组件的 ID 设置为 submit 和 reset。而如果想对提交、重置进行特殊的处理，则可以定义其它 ID 名称的操作按钮进行相应的 JS 编程(如想提交其它特定的数据等)

4.6.2. 属性介绍

- id：表单的惟一标识
- name：HTML 标准属性
- method：HTML 标准属性，缺省采用的是 post 方法
- enctype：HTML 标准属性，如果需要文件上传，则该属性值应设置为：multipart/form-data，缺省情况下不对该属性进行设置

- action：HTML 标准属性
- target：HTML 标准属性
- onsubmit：HTML 标准属性，其中使用 this 表示当前的 form 标签对象

4.6.3. 配置示例

```
<emp:form id=”submitForm” action=”signOn.do”>
    … …
    <emp:button id=”submit” label=”提交”/>
</emp:form>
```

生成的缺省 JS 方法如下：

```
function doSubmit(button){
    var form = document.getElementById('submitForm');
    var result = page.dataGroups.dataGroup_in_formsubmitForm.checkAll();
    if(form && result){
        page.dataGroups.dataGroup_in_formsubmitForm.toForm(form);
        form.submit();
    }
};
```

4.7. emp:page标签

emp:page 标签比较简单，而且功能也比较单一，其作用就是为页面提供对 EMP 其它标签的自动注册功能。如果有哪个 EMP 标签不想被注册，那么可以将该标签放到 emp:page 标签的外面，反之亦然。

除此之外，emp:page 还可以设置是否在页面载入时就自动的启动注册的工作，该功能是通过 emp:page 标签中的 autoLoad 属性进行定义，在缺省情况下是在载入时就启动注册的工作。

因此，对于一般的页面而言，只需要使用 emp:page 标签将页面中的所有内容包含进来，就可以自由的使用各种“注册”后的 JS 对象了。

5. JS工具类(EMPTools)

5.1. addEvent

addEvent 方法是用于对某个 HTML 页面元素添加特定的事件，使用的方法如下：

```
EMPTools.addEvent(element, "blur", onBlurEvent, window)
```

其中的四个参数分别是：

- element：需要添加事件的页面元素对象
- eventName：需要添加的事件名称，如 blur、click、change(不需要带前缀 on)
- method：事件触发后调用的 JS 方法(需要注意的是这里指的是 JS 方法，则不是 JS 方法的名称，如上述例子必须保证在执行这段 JS 之前，在页面中有一个 onBlurEvent 的方法)
- host：调用 JS 方法的对象，即所调用的 JS 方法中使用 this 所代表的对象

5.2. getWindowOpener

获得特定弹出窗口的父窗口(同时兼容 window.open 和 window.showModelDialog)

其中的参数是：

- winObj：特定弹出窗口的 window 对象，如果该参数为空，则表示获得当前弹出窗口的父窗口对象

5.3. openWindow

打开当前页面的弹出窗口(同时兼容 window.open 和 window.showModelDialog)

其中的参数是：

- url：弹出窗口的 url
- winId：弹出窗口的 id，只在 window.open 时才起作用
- popparam：弹出窗口的参数设置，如大小、位置等等，缺省情况下的参数是：height=400, width=600, top=120, left=200, toolbar=no, menubar=no, scrollbars=yes, resizable=no, location=no, status=no

5.4. trim

将字符串前后空格去除

其中的参数是：

- str：需要去除空格的字符串

5.5. getByteLength

计算字符串的长度，用于计算 HTML 本身无法判断的双字节字符长度

其中的参数是：

- str：需要计算长度的字符串

5.6. encodeURI

对 url 进行规整并使用 js 的 encodeURI 进行编码

其中的参数是：

- url：需要 encode 的字符串

通常的使用方式是：

```
var url = '<empext:url url="getSLogUpdatePage.do"/>?'+paramStr;
url = EMPTools.encodeURI(url);
```

5.7. setWait

在特定页面上显示请稍候的标识

通常的使用环境是页面中使用 AJAX 请求进行部分的业务处理时，通过显示请稍候的标识可以让用户比较容易的知道当前的处理是否已经完成

其中的参数是：

- windowObj：需要显示请稍候标识的页面 window 对象，缺省情况下是指当前的页面

5.8. removeWait

取消特定页面上请稍候的标识

其中的参数是：

- windowObj：需要取消的页面 window 对象，缺省情况下指的是当前的页面

5.9. ajaxRequest

使用 AJAX 进行异步的请求

其中的参数是：

- method：异步请求的方式：GET\POST
- url：异步请求的 url
- callback：异步请求和回调方法所需要的参数列表。其中 success 参数是必须的，代表的是回调的方法或方法的名称。除了 success 之外还有几个参数，分别是：
  - isJSON：用于说明 AJAX 请求返回的字符串是否是一个合法的 JSON 串，如果是的话，则将返回的字符串转成 JS 对象并传递给回调方法，否则直接将返回的字符串传递给回调方法；如果 isJSON 为 true，但转成 JS 对象的过程中出错，则将该属性更改为 false 并传递给回调方法。缺省情况下，该属性为 true
  - handleError：是否自行处理异常信息，如果需要自行处理异常信息，则将异常信息放到 errorMessage 属性中传递给回调方法。缺省情况下，该属性为 false

使用的方法是：

```
function doReturnMethod(json, callback){
    var name = json.stockName;
    Declare.stock_name._setValue(name);
}


var    url    =    "<empext:url    url='getStockName.do'/>?stockId="+
Declare.stock_code._getValue();


var callback = {
    success : "doReturnMethod",
    isJSON : true
```

```
};
EMPTools.ajaxRequest('GET', url, callback);
```

其中的回调方法拥有两个参数，分别是

- json：异步请求返回的结果
- callback：异步请求的参数列表以及可能拥有的错误信息

5.10. log

该方法用于在调试状态下显示调试信息。对于正常的页面，可以通过引用 log.js 和 firebug-lite-compressed.js 这两个 JS 文件来表示当前处于调试状态中，因此该方法也只在这两个 JS 文件都引用的情况下才能显示出相应的调试信息

其中的参数是：

- type：日志的类型
- level：日志的级别，如果级别为 4，则使用 alert 方式弹出日志的信息
- msg：日志的信息
- e：异常的对象，如果该属性不为空，则向外抛出该异常

5.11. message

JS 方法返回的消息对象(通常用在校验中，具体属性的作用在不同的模块中可能有不同的含义)

其中的参数是：

- result：操作结果(true 表示成功；false 表示失败。缺省情况下表示成功)
- message：消息对象的具体内容(如失败时表示失败的信息。缺省情况下为空)
- level：消息的级别(0：只在标签的后面显示信息；1：将信息使用 alert 弹出。缺省情况下只显示)
- control：消息对于表单提交操作的控制作用(0：不可忽略；1：可忽略；2：不需要检查。缺省情况下是不可忽略的)

目前用到消息对象的主要有两个地方：

- 数据类型校验，其中 message 属性在成功时表示所需要的真实值或显示值，而在失败时表示失败的信息

- 页面的业务校验接口，其中的 message 属性只在失败时表示失败信息，其它情况不处理

5.12. addClass

增加某个 HTML 元素的指定 css 样式

其中的参数有：

- el：指定的 HTML 元素对象
- className：所增加的 CSS 样式名称

5.13. removeClass

删除某个 HTML 元素的指定 css 样式

其中的参数有：

- el：指定的 HTML 元素对象
- className：所要删除的 CSS 样式名称

5.14. getScrollPos

获得页面边缘与容器(比如窗口)之间的间距

- 返回值中的"x"属性 ：左边缘间距(横向)
- 返回值中的"y"属性 ：上边缘间距(纵向)

5.15. setInnerText

设置某个 HTML 元素的 innerText

其中的参数有：

- el：指定的 HTML 元素对象
- text：所要设置的文本内容

5.16. getElementById

获得 HTML 元素中包含的某个标签名称的指定 id 对象

其中的参数有：

- el：指定的 HTML 元素对象
- id：指定的 ID
- tagName：指定的标签名称

5.17. seeObject

得到某个对象中的所有属性(或方法)，只是个简单的方法，只能获得一个层次的数据，无法递归

其中的参数有：

- obj：所要展现的对象

例如：obj 中有两个属性：param1,param2 和一个方法 fun1，那么该方法返回的字符串为：“param1=111;param2=aaaa;fun1=function(){};”这样的一段字符串。主要的用途在于调试时能够比较方便的得到对象中所包含的属性值

5.18. getChildrenByTagName

获得指定 HTML 元素中包含的所有指定标签名称的对象

其中的参数有：

- el：所指定的 HTML 元素
- tagName：所指定的标签名称

5.19. getParam

得到 url 中指定参数的值

其中的参数有：

- url：要取值的 url

- param：所指定的参数名称

5.20. setParam

对 url 中指定的参数进行设值

其中的参数有：

- url：要设值的 url
- param：所指定的参数名称
- value：需要设置的值