表单序列化、规则分离下的复杂表单计算

这篇文章发布于 2013年01月31日，星期四，20:05，归类于 jQuery相关。 阅读 56783 次, 今日 7 次 20 条评论

一、关于表单计算

电商螃蟹般横行，类似购物车涉及计算的表单频频照面。

这些表单可能比较简单，也可能同时涉及数量变动、运费、保价等，计算会比较复杂。

二、你是如何实现?

有一种常见实现是这样子的：单价、数量输入框、小计价格与商品id关联，类似下面代码：

```
<span id="unitPrice_110">400</span>元      <input id="unitNumber_110" name="unitNumber_110
" value="1" />      小计：<span id="unitTotal_110">400</span>元
<span id="unitPrice_111">50</span>元       <input id="unitNumber_111" name="unitNumber_111
" value="2" />      小计：<span id="unitTotal_111">100</span>元
```

然后有个全局输入的id商品数组：

```
var arrIdGoods = [110, 111];
```

然后，遍历id进行事件绑定①，当数量改变时候，从DOM元素获得单价然后进行计算以及修改后数值的呈现：

```
$.each(arrIdGoods, function(i, id) {
    $("#unitNumber_" + id).bind("input", function() {
        // 下面标红为DOM元素取值
        $("#unitTotal_" + id).html($("#unitPrice_" + id).html() * this.value);
    });
```

```
});
```

//zxx: ① 除了遍历数组id进行事件绑定，还有直接遍历输入框元素进行事件绑定的（通过输入框上类似 `data-id` 匹配其他元素）

以上实现也是很OK的，至少很灵活。但是，不足在于，计算的成本会随着计算逻辑的增加而暴增！

举个例子：
商品数量是4个的时候，免运费；超过4个要增加运费。价格100元以上免运费，总价小于100元收6元运费；可以使用积分抵扣；可以使用优惠券；还有保险……此时你该如何应对？

该不会是找到对应的一个一个的DOM元素，获取里面的值，然后再在JS中处理逻辑，进行计算的吧！！？？

说不定你很享受这个，展现我逻辑思维能力的时候到了，而且，我还可以————————————因为这个加班哦！

即使历经艰难，终于实现了前前后后改动数值的无刷新改动。由于计算逻辑在JS中，后期维护很闹心，例如，一旦计算规则修改，势必又要去折腾JS文件。又由于计算逻辑和DOM处理混在一起，代码什么就是个妙脆角——一咬就碎！而且，你这代码只能用在这个表单上，换个其他表单，你又要重新折腾一遍，唉，怪不得雀巢咖啡在土豆上狂打广告~~

————————低调的分隔线————————

有必要向PM反映：“丫的这里的计算太复杂了，能不能改改哈”！

如果你深知，让PM做修改比你为绑架的PM交赎金还难，上面一句话就当我没说过。
如果你向PM吐槽了，PM还真就做了修改，如：计算分布走，降低耦合。那么恭喜你🤝，你中头奖啦！
如果你的建议被北京的大雾给完全遮蔽了，兄弟，你有两条路可以走，一是：我加班我喜欢，啦啦啦啦~~二是，眼珠往下扫50像素，继续阅读本文！

三、分离的表单计算

我喜欢看美剧比喜欢看韩剧多一点。因为韩剧动不动就三角恋、要么四角恋，好闹心的来；而美剧，只要有恋，就去滚床单了，多简单明快啊，我喜欢。

实现功能也是如此，搞来搞去的，想想都烦。你说这表单计算，计算逻辑和DOM呈现混在一起，页面一复杂，计算逻辑一多，代码还不乱套！有必要降低耦合，分离！

如何分离？
很简单，计算规则独立出来，就在页面上，由后台或编辑控制；作为前端的我们要做的事情就一个——

→ 套用分离出来的计算规则，呈现结果！

哈哈，是不是有兴趣啦。

然而，就像切猪肉一样，卖肉的师傅三下五除二就把猪皮分离出来的，看似简单，真正我们动手的时候，说不定心中就会大火：师傅，你怎么能把你的猪皮用502胶粘在肉上呢！

分离作用

分离的好处除了上面降低耦合，简化处理，增加可维护性；还有一个莫大的好处——实现表单计算的通用方法。

这个通用方法才是本文最重要的干货，可以让你以后所有的表单计算都像弹棉花那样轻松。

四、爱我，别走~~

只吆喝，没干货，观众会走的。因此，打破顺序，先展示demo。

您可以狠狠地点击这里：表单序列、规则分离下的购物车表单计算demo

demo基于jQuery实现，可以修改数量、删除、保价以及积分使用。截图缩略图如下：

篇幅原因，源代码为展示，您可以使用前端工具，或右键→页面源代码查看。

五、表单计算通用方法、分离规则的特定格式

//zxx: 本文的表单计算通用方法基于jQuery, 因此，其他JS框架下该方法可能受限，若要适用，您可能需要做一定的语法上的修改。

方法名以及参数如下：

```
$.calculate(form, rule);
```

`form` 表示表单元素（非 `form` 元素也可以）； `rule` 表示计算规则。

该方法很短很简单，连注释等就几十行，压缩后不足1K，下面就是压缩后（节约篇幅）的完整代码：

```
/*! 基于分离规则的复杂表单计算方法
 * by zhangxinxu(.com) 2013-01-31
*/
(function(a){a.calculate=function(c,e){if(!e){return}c=a(c);var d=c.data("serialize")||{};
var b={},f=function(g){return typeof g=="number"&&(g||g===0)};a.each(d,function(g){d[g]=0}
);c.find(":input").each(function(){var i,g=this.name,h=this.type;if(g){i=a(this).val();if(
/radio|checkbox/.test(h)){if(d[g]){return}if(this.checked&&!this.disabled){d[g]=i*1||i}els
e{d[g]=0}}else{if(!i||this.disabled){i=0}d[g]=i*1||i}}});c.data("serialize",d);b=a.extend(
{},d);a.each(e,function(i,g){var h=a.isFunction(g)?g.call(b):g;if(!f(h)){h=0}b[i]=h});a.ea
ch(e,function(j,g){var i=/^\w|\[|\:/.test(j)?a(j):a("#"+j),h=b[j]||(a.isFunction(g)?g.call
(b):g)||0;if(f(h)&&i.length){!b[j]&&(b[j]=h);h=String(Math.round(h*100)/100).replace(/\.00
/,"");i.each(function(){if(/^input$/i.test(this.tagName)){a(this).val(h)}else{a(this).html
(h)}})}})}})(jQuery);
```

为了方法复用，虽然方法很小，我还是独立在一个JS文件中了，您可以狠狠地点击下载：jquery-calculate.js

或者像下面这样调用：

```
<script src="http://img3.douban.com/js/packed_jquery.min6301986802.js"></script>
<script src="http://http://www.zhangxinxu.com/study/201301/jquery-calculate.js"></script>
```

分离计算规则的特定格式

为了匹配`$.calculate()`表单计算，`rule`参数对应分离的规则对象需要有一定的格式，其实很简单，如下示意：

```
var rule = {
    "计算结果元素的id或选择器": "计算规则" // 规则常量则直接数值，规则动态则方法
};
```

套用一开始的例子：

```
<span id="unitPrice_110">400</span>元      <input id="unitNumber_110" name="unitNumber_110
" value="1" />      小计：<span id="unitTotal_110">400</span>元
<span id="unitPrice_111">50</span>元       <input id="unitNumber_111" name="unitNumber_111
" value="2" />      小计：<span id="unitTotal_111">100</span>元
```

则：

```
var rule = {
   unitTotal_110: function() {
       return 400 * this.unitNumber_110;
   }
   unitTotal_111: function() {
       return 50 * this.unitNumber_111;
   }
};
```

上面代码标红的部分就是动态内容，其对应相应表单的`name`值。

我们表单提交时候，提交的数据都是根据表单元素的name值索引的。这里的实现也是类似这个原理。`$.calculate`方法做了两件事情：

1. 表单序列化form元素，得到`name - value`键值对对象。
2. 解析`rule`对象，`this.name`用第一步中`name`对应的`value`替换；然后返回计算结果，值呈现。

套用上面的例子：

1. 表单序列化，得到：

   ```
   {
      unitNumber_110: 1,
      unitNumber_111: 2
   }
   ```

   。

2. 解析`rule`对象，赋值：

   ```
   $("#unitTotal_110") = 400 * this.unitNumber_110 = 400 * 1 = 400
   $("#unitTotal_111") = 50 * this.unitNumber_111 = 50 * 2 = 100
   ```

over!

套用demo更详尽的展示

demo中一些计算规则如下：

> 1. 运费起步价6元，当货物总数大于4个时候，每多1个商品，运费增加1元
> 2. 当购物车没有商品时候，即使勾选“报价”，报价费用也不在总价中，总价为0
> 3. 积分兑换比例为100:1, 也就是100积分可以抵实际的1元人民币

下面的代码为demo中规则的完整代码：

```
var oRule = {
    // 商品1小计价格
    subTotal_1: function() {
        return 400 * this.unitNumber_1;
    },
    // 商品2小计价格
    subTotal_2: function() {
        return 50 * this.unitNumber_2;
    },
    // 运费
    transFare: function() {
        var overflow = this.unitNumber_1 + this.unitNumber_2 - 4;
        if (overflow <= 0) {
            overflow = 0;
        }
        return 6 + overflow;
    },
    // 积分等同的价钱
    trueValue: function() {
        return this.storedScore / 100;
    },
    // 总价
    allTotal: function() {
        if (this.subTotal_1  + this.subTotal_2 === 0) return 0;
        return this.subTotal_1  + this.subTotal_2 + this.protect + this.transFare - this.t
rueValue;
    }
};
```

对oRule对象的一些说明：

- `"subTotal_1"`，`"subTotal_2"`，`"transFare"`，`"trueValue"`，`"allTotal"`，分别为下图圈中元素的id，也就是所有DOM中数值改变的元素。

  注：1. `rule` 对象关键字不仅可以是元素id，也可以是 `id选择器` 或 `class选择器`，如 `"#subTotal_1"`.

  2. 元素可以是普通元素，也可以是表单控件，例如 `hidden` 类型隐藏文本域，你可以把计算后的值藏在其中，表单提交时候就用得上了。

- `this` 是必须的也只能写做 `this`，不要以常规的想法理解这里的 `this`。
- `this` 后面的属性值，不仅可以是表单元素的 `name` 值，还可以是规则对象中的关键字——表示直接调用已经计算出来的值，在计算总价的时候很好用。不过需要注意的是，使用的关键字(eg. `"subTotal_1"`)不能与任何表单元素的 `name` 值冲突！

补充：有同行反映了多次依赖调用的问题。目前，在无序状态下（如后调用的在前），只能1次依赖；有序状态下，2次依赖，2次为上限。如要实现多次依赖，需要在书写上做一点修改：例如：

```
var rule = {
    demo1: 5,
    demo2: function() {
        return this.demo1 +5;
    },
    demo3: function() {
        return this.demo2 +5;
    }
};
```

改成：

```
var rule = {
    demo1: 5,
    demo2: function() {
        return this.demo1 +5;
    },
    demo3: function() {
        return this.demo1 + 5 +5;
    }
};
```

- `rule`对象中，如这里的`oRule`，关键字的顺序你可以随意排列，例如，把总价放在第一个。`$.calculate()`已做了处理。
- 无需担心`this.`后面的`name`元素不存在的情况，`$.calculate()`已做了处理。总之，一句话，只要关心运算逻辑即可。
- `rule`对象关键字还可以是与元素无相干的数据。例如，每个计算规则都用到一个常量，则可以：

```
oRule = {
    aUsefulNumber: 3.1415;
    ".result": function() {
        return this.aUsefulNumber* this.goodsNumber;
    }
};
```

只要`aUsefulNumber`与其他`name`或关键字不冲突就可以了。

- `this.name`的值格式任意，如果是数值字符串，则数值返回；如果是普通字符串，依然是普通字符串。例如，如下HTML：

```
<select name="select">
    <option value="test">字符串</option>
    <option value="100">数值字符串</option>
</select>
```

则根据下拉选项的不同，`this.select`的实际值为字符串`"test"`或数值`100`. 如果文本框值为空，或者单复选框为选择，则返回的是数值0. 这个需要注意，我们可以使用是否值为`0`判断一个单选框组是否有选中，此时不能设置任意一个单选框的value为`"0"`，否则判断就会冲突。

- 计算值呈现的时候，保留2为有效数字，如果最后两位为`00`，则直接取整。

有此可见，`rule`对象很灵活，容错性很强，可以满足各类表单计算需求。

事件的触发与计算结果的呈现

您可以想demo一样，直接写个贯穿的方法，而不需要像以前一样，很苦逼的为每个元素专门写事件些逻辑。例如：

```
eleForm.find("input").bind("input", function() {
    $.calculate(eleForm, oRule);
});
```

over啦！哈哈，简单地超乎想象吧~~

六、另外的实例：涉及单选组的表单计算

本例子展示动态计算规则的应用。

您可以狠狠地点击这里：含单选组的动态结算表单demo

截图如下：

点击不同的单选按钮，应用不同的计算规则，规则对象代码如下：

```
var oRule = {
    ".totalPrice": function() {
        // 单选组的值为返回的是数值
        switch (this.rule) {
            case 1: {
                return this.person * 20;
                break;
            }
            case 2: {
                return this.expense * 0.12;
                break;
            }
            case 3: {
                return this.order * 0.15 + (this.expense - this.order) * 0.1;
                break;
            }
            default: {
                return 0;
            }
        }
    }
};
```

恩，关于`oRule`对象要说的上面一段都说了，上面标红的`".totalPrice"`是进一步提醒－关键字可以是选择器。例如，上面demo的总价不仅在元素上呈现，还赋给了一个`className`为`totalPrice`的隐藏文本框，以满足实际需求。

七、结语

我平时项目不是使用的jQuery, 因此，由于缺少足够的实践测试，因此，势必有些小bug之类。如果承蒙你试用，同时发现了问题，欢迎指出（评论形式或邮箱zhangxinxu$zhangxinxu.com或微博http://weibo.com/zhangxinxu），不甚感谢！

一不留神，啪啪啪啪写了这么多，我要休息下了……

（本篇完）　// 想要打赏？点击这里。有话要说？点击这里。

« 我对原型对象中this的一个懵懂错误认识　　github上html5shiv项目readme.md部分的翻译 »

猜你喜欢

- CSS样式分离之再分离
- 翻译-不同CSS技术及其CSS性能
- 精简高效的CSS命名准则/方法
- CSS流体(自适应)布局下宽度分离原则
- jQuery html5Validate基于HTML5表单验证插件
- 翻译：关于表单每个设计师都必须知道的10件事
- HTML5 number类型文本框step属性的验证机制
- 残忍：IE10 IE7-IE9 type=email的完全抛弃
- 伪元素表单控件默认样式重置与自定义大全
- 小tip: CSS动态实现文本框清除按钮的隐藏与显示
- jQuery诞生记-原理与机制

标签：　dom, jQuery, 分离, 表单, 计算

发表评论（目前20条评论）

名称 (必须)

邮件地址(不会被公开) (必须)

网站

提交评论

1. 匿名说道：
2017年02月21日 17:18

修改数量的时候绑定.serialize()，博主的方法滞后了，数据对不上

回复

2. 匿名说道：
2017年02月21日 17:05

不知道怎么表达，总是滞后，难道是执行效率的问题?

```
// 商品数量
p_pn: function() {
return this.unit_num;
},

//购物车
function ajax_editCar(){
$.ajax({
type: 'get',
url: 'ajax.php',
data: $("#car_form").serialize()+"&now="+new Date().getTime(),
success: function(data) {
// your code
//alert(data);
}
});
}
```

同步计算没有问题，传递表单post的时候总是滞后一步。
数量显示3的时候，post表单是2

回复

3. 吴英凤说道：
2016年06月29日 14:27

另外看鑫哥的源码感觉很多地方写的都很精辟，要不你有时间把那些精辟之处单独说说呗

回复

4. 吴英凤说道：
2016年06月29日 14:25

form.data("serialize")鑫哥这个是怎么来的呀

回复

5. 猫五号说道：
2013年03月20日 16:33

这货是MVVM

回复

6. **edielei**说道：
2013年02月27日 02:47

请注意安全性。。别让用户使用浏览器开发人员工具给修改了价格，还能以这个价格支付就行！

回复

皮皮阳说道：
2013年06月6日 10:47

价格是存在数据库里的，这个只是view层的表现，即便修改前端价格数据也不会造成影响

回复

7. **r23**说道：
2013年02月17日 16:43

onpropertychange类似于oninput IE专用，如果撸主是故意不用，请说下原因

回复

张 鑫旭说道：
2013年02月19日 10:10

@r23 才疏学浅，学习了！

回复

8. **baoeni**说道：
2013年02月6日 05:10

三个小建议：

1. 可以用value.toFixed(2)来替代Math.round(value * 100) / 100

2. 可以用jquery本身自带的form.serializeArray函数

3. 可以扫描rule里的name，只有当这些elements变化的时候才触发这个rule的计算，这样效率就高了。

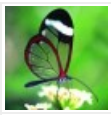
回复

张 鑫旭说道：
2013年02月6日 10:26

@baoeni 1. value.toFixed(2)并不是标准的四舍五入，而且replace执行的时候会变成位数很多的浮点型；2. form.serializeArray有三个问题，一是数据格式，还要额外map处理，多一次循环；二是只能获得当前form的内容，对于删除后的元素没有保留处理，可能会造成计算式的报错；三是一些特殊处理，例如元素disabled的时候也要取值（作为0处理），serializeArray则直接缺失该关键字。3. rule中的name有可能不是元素，至于元素什么时候触发计算是有你自己控制的，你可以把input触发改成change触发。

回复

9. **lovelucy**说道：
2013年02月5日 16:03



@ax1.1 类似的MVC框架太多了，不过都不成熟
比如 http://documentcloud.github.com/backbone
比如 http://angularjs.org/
比如 http://emberjs.com/

不同应用场景和业务复杂度，没有银弹啊，选择困难。。。

回复

10. 疯子说道：
2013年02月4日 09:05

学习了，以前也做过购物车不过没有博主想的这么全。

回复

11. **ax1.1**说道：
2013年02月1日 15:35

这是个MVVM模式的实现嘛
建议撸主看下 http://knockoutjs.com

回复

12. **devan5说道：**

2013年02月1日 15:18

想知道如何解决在计算中的多次依赖.
比如算出总价后,计算得一次加税,再计算得一次加税,若再有几次依赖上一次计算..

附上一个小测试:

```
var opts = {

demo2: function(){
return this.demo1 + 5;
},

demo3: function(){
return this.demo2 + 5;
},

demo1: function(){
return 5;
}
};

$.calculate($(“form”),opts);

5 // demo1
10 // demo2
5 // demo3
```

结果不是期望的值.
但调整 prop 属性为 demo1,demo2,demo3 结果正确.

回复

> 张 鑫旭说道：
>
> 
>
> 2013年02月1日 16:27
>
> @devan5 目前无序状态只能满足1次依赖，有序状态可以满足2此依赖，2次是极限。若要实现你要的效果，需要在书写上做一点调整，例如下面这样：
>
> ```
> var opts = {
> demo2: function(){
> return this.demo1 + 5;
> },
>
> demo3: function(){
> return this.demo1 + 5 + 5;
> },
>
> demo1: function(){
> return 5;
> }
> };
>
> $.calculate($(“form”),opts);
> ```
>
> 回复

13. **F1y4L0ve说道：**

2013年02月1日 10:51

可能我考虑的太极端了，既然是个通用算法，应该考虑大数据量的情况。
这个算法里，数据量加大，有可能影响效率的地方有三个。

1.Form的序列化
2.对无影响的数据重复计算。如果算法复杂，这个不能不考虑。
3.计算结果刷新，和dom打交道。

一个想法
1，只序列化一次form，页面基础数据有变化，只调整相应数据。
2，设计计算优先级以及影响链表，只对影响结果计算。
3，根据影响链表，只调整受影响结果显示。

优势自然是速度，缺点是页面保持一个自定义的序列化数据，二是序列化规则稍微比较麻烦。

之所以介意效率问题，是因为以前项目确实有个比较大的表单，js计算超慢。

以上没进行过测试，有点儿想当然，如果有什么问题，见谅。

回复

14. **F1y4L0ve说道：**

2013年02月1日 09:42

感觉无序计算还是蛮危险的，最好序列化时加个优先级。另外动一下就全部重新计算，数据多了，效率也是个问题。

回复

> 张 鑫旭说道：
>
> 
>
> 2013年02月1日 09:55
>
> @F1y4L0ve ① 无需担心无序，方法已经做了特殊的处理，这可是方法的亮点哦！
> ② 重新计算是在数据层面，效率几乎可以忽略不计。顶多0毫秒和1毫秒的差距。
>
> 回复

15. **cangzhang说道：**

2013年01月31日 22:46

好文！楼主很用心！

回复

最新文章
» 常见的CSS图形绘制合集
» 粉丝群第1期CSS小测点评与答疑
» 分享三个纯CSS实现26个英文字母的案例
» 小tips: 纯CSS实现打字动画效果
» CSS/CSS3 box-decoration-break属性简介
» CSS :placeholder-shown伪类实现Material Design占位符交互效果
» 从天猫某活动视频不必要的3次请求说起
» CSS vector-effect与SVG stroke描边缩放
» CSS ::backdrop伪元素是干嘛用的?
» 周知：CSS -webkit-伪元素选择器不再导致整行无效

今日热门
» 常见的CSS图形绘制合集 (193)
» 未来必热：SVG Sprite技术介绍 (120)
» 粉丝群第1期CSS小测点评与答疑 (115)
» HTML5终极备忘大全（图片版+文字版） (93)
» 让所有浏览器支持HTML5 video视频标签 (86)
» Selectivizr-让IE6~8支持CSS3伪类和属性选择器 (82)
» CSS3下的147个颜色名称及对应颜色值 (80)
» 视区相关单位vw, vh..简介以及可实际应用场景 (76)
» 写给自己看的display: flex布局教程 (76)
» 小tips: 纯CSS实现打字动画效果 (76)

今年热议
» 《CSS世界》女主角诚寻靠谱一起奋斗之人 (76)
» 不借助Echarts等图形框架原生JS快速实现折线图效果 (64)
» 看，for..in和for..of在那里吵架！ (60)

» 是时候好好安利下LuLu UI框架了！ (47)
» 原来浏览器原生支持JS Base64编码解码 (35)
» 妙法攻略：渐变虚框及边框滚动动画的纯CSS实现 (33)
» 炫酷H5中序列图片视频化播放的高性能实现 (31)
» CSS scroll-behavior和JS scrollIntoView让页面滚动平滑 (30)
» windows系统下批量删除OS X系统.DS_Store文件 (26)
» 写给自己看的display: flex布局教程 (26)

猜你喜欢

- CSS样式分离之再分离
- 翻译-不同CSS技术及其CSS性能
- 精简高效的CSS命名准则/方法
- CSS流体(自适应)布局下宽度分离原则
- jQuery html5Validate基于HTML5表单验证插件
- 翻译：关于表单每个设计师都必须知道的10件事
- HTML5 number类型文本框step属性的验证机制
- 残忍：IE10 › IE7-IE9 type=email的完全抛弃
- 伪元素表单控件默认样式重置与自定义大全
- 小tip: CSS动态实现文本框清除按钮的隐藏与显示
- jQuery诞生记-原理与机制