Table of Contents

誉道科技

基础规范

誉道科技技术部规范文档

主要包含技术部开发过程中涉及的开发流程、设计规范、代码规范、测试规范等说明。

命名规则

注释规范

HTML规范

js规范

css规范

驼峰式命名法介绍

Pascal Case 大驼峰式命名法：首字母大写。例如：StudentInfo、UserInfo、ProductInfo

Camel Case 小驼峰式命名法：首字母小写。例如：studentInfo、userInfo、productInfo

文件命名

全部采用小写方式， 以下划线分隔。

例：my_project_name

变量命名

采用 小驼峰式命名方法。例如：studentInfo、userInfo、productInfo

函数命名

命名方式 : 小驼峰方式
命名规则 : 前缀为动词

动词	含义	返回值
can	判断是否可执行某个动作 (权限)	函数返回一个布尔值。true：可执行；false：不可执行
has	判断是否含有某个值	函数返回一个布尔值。true：含有此值；false：不含有此值
is	判断是否为某个值	函数返回一个布尔值。true：为某个值；false：不为某个值
get	获取某个值	函数返回一个非布尔值
set	设置某个值	无返回值、返回是否设置成功或者返回链式对象

推荐：

```
//是否可阅读
function canRead(){
    return true;
}

//获取姓名
function getName(){
```

```
    return  this.name
}
```

常量命名

命名方法: 全部大写
命名规范: 使用大写字母和下划线来组合命名，下划线用以分割单词。
推荐：

```
var MAX_COUNT = 10;
var COUNT =1;
```

Pascal Case 大驼峰式命名法：首字母大写。例如：StudentInfo、UserInfo、ProductInfo

Camel Case 小驼峰式命名法：首字母小写。例如：studentInfo、userInfo、productInfo

全部采用小写方式， 以下划线分隔。

例：my_project_name

命名方式

采用 小驼峰式命名方法。例如：studentInfo、userInfo、productInfo

命名方式 : 小驼峰方式
命名规则 : 前缀为动词

动词	含义	返回值
can	判断是否可执行某个动作 (权限)	函数返回一个布尔值。true：可执行；false：不可执行
has	判断是否含有某个值	函数返回一个布尔值。true：含有此值；false：不含有此值
is	判断是否为某个值	函数返回一个布尔值。true：为某个值；false：不为某个值
get	获取某个值	函数返回一个非布尔值
set	设置某个值	无返回值、返回是否设置成功或者返回链式对象

推荐：

```
//是否可阅读
function canRead(){
    return true;
}

//获取姓名
function getName(){
    return  this.name
}
```

命名方法: 全部大写
命名规范: 使用大写字母和下划线来组合命名，下划线用以分割单词。
推荐：

```
var MAX_COUNT = 10;
var COUNT =1;
```

单行注释

单独一行：//(双斜线)与注释文字之间保留一个空格

- 在代码后面添加注释：//(双斜线)与代码之间保留一个空格，并且//(双斜线)与注释文字之间保留一个空格。
- 注释代码：//(双斜线)与代码之间保留一个空格。

> 推荐：

```
// 调用了一个函数； 1)单独在一行
setTitle();
var maxCount = 10; // 设置最大量； 2)在代码后面注释
// setName();  //3)注释代码
```

多行注释

若开始(/*和结束(*/)都在一行，推荐采用单行注释

若至少三行注释时，第一行为/*，最后行为*/。

推荐：

```
/*
代码执行到这里后会调用setTitle()函数
setTitle()：设置title的值
*/
setTitle();
```

函数(方法)注释

函数(方法)注释也是多行注释的一种，但是包含了特殊的注释要求
语法：

```
/**
* 函数说明
* @关键字
*/
```

常用注释关键字

注释名	语法	含义	示例	是否

注释名	语法	含义	示例	必填
@param	@param 参数名 {参数类型} 描述信息	描述参数的信息	@param name {String} 传入名称	是
@return	@return {返回类型} 描述信息	描述返回值的信息	@return {Boolean} true:可执行;false:不可执行	是
@author	@author 作者信息 [附属信息：如邮箱、日期]	描述此函数作者的信息	@author 张三 2015/07/21	否
@version	@version XX.XX.XX	描述此函数的版本号	@version 1.0.3	否
@example	@example 示例代码	@example setTitle('测试')	如下	否

推荐 :

```
/**
 - 合并Grid的行
 - @param grid {Ext.Grid.Panel} 需要合并的Grid
 - @param cols {Array} 需要合并列的Index(序号)数组；从0开始计数，序号也包含。
 - @param isAllSome {Boolean} ：是否2个tr的cols必须完成一样才能进行合并。true：完成一样；fa
lse(默认)：不完全一样
 - @return void
 - @author polk6 2015/07/21
 - @example
*/

function
 mergeCells(grid, cols, isAllSome) {
    // Do Something
}
```

单独一行：//(双斜线)与注释文字之间保留一个空格

- 在代码后面添加注释：//(双斜线)与代码之间保留一个空格，并且//(双斜线)与注释文字之间保留一个空格。
- 注释代码：//(双斜线)与代码之间保留一个空格。

> 推荐：

```
// 调用了一个函数； 1)单独在一行
setTitle();
var maxCount = 10; // 设置最大量； 2)在代码后面注释
// setName();  //3)注释代码
```

若开始(/*和结束(*/)都在一行，推荐采用单行注释

若至少三行注释时，第一行为/*，最后行为*/。

推荐 :

```
/*
代码执行到这里后会调用setTitle()函数
setTitle()：设置title的值
*/
setTitle();
```

函数(方法)注释也是多行注释的一种，但是包含了特殊的注释要求
语法：

```
/**
* 函数说明
* @关键字
*/
```

常用注释关键字

注释名	语法	含义	示例
@param	@param 参数名 {参数类型} 描述信息	描述参数的信息	@param name {String} 传入名称
@return	@return {返回类型} 描述信息	描述返回值的信息	@return {Boolean} true:可执行;false:不可执行
@author	@author 作者信息 [附属信息：如邮箱、日期]	描述此函数作者的信息	@author 张三 2015/07/21
@version	@version XX.XX.XX	描述此函数的版本号	@version 1.0.3
@example	@example 示例代码	@example setTitle('测试')	如下

推荐 :

```
/**
 - 合并Grid的行
 - @param grid {Ext.Grid.Panel} 需要合并的Grid
 - @param cols {Array} 需要合并列的Index(序号)数组；从0开始计数，序号也包含。
 - @param isAllSome {Boolean} ：是否2个tr的cols必须完成一样才能进行合并。true：完成一样；fa
lse(默认)：不完全一样
 - @return void
 - @author polk6 2015/07/21
 - @example
*/

function
 mergeCells(grid, cols, isAllSome) {
    // Do Something
}
```

文档规范

使用 HTML5 的文档声明类型：`<!DOCTYPE html>`

- DOCTYPE标签是一种标准通用标记语言的文档类型声明，它的目的是要告诉标准通用标记语言解析器，它应该使用什么样的文档类型定义（DTD）来解析文档。
- 使用文档声明类型的作用是为了防止开启浏览器的怪异模式。
- 没有DOCTYPE文档类型声明会开启浏览器的怪异模式，浏览器会按照自己的解析方式渲染页面，在不同的浏览器下面会有不同的样式。
- 如果你的页面添加了<!DOCTYP>那么，那么就等同于开启了标准模式。浏览器会按照W3C标准解析渲染页面。

脚本加载

说到js和css的位置，大家应该都知道js放在下面，css放在上面。

> 所有浏览器中推荐:

```
<html>
  <head>
    <link rel="stylesheet" href="main.css">
  </head>
  <body>
    <!-- body goes here -->

    <script src="main.js" async></script>
  </body>
</html>
```

> 只兼容现代浏览器推荐:

```
<html>
  <head>
    <link rel="stylesheet" href="main.css">
    <script src="main.js" async></script>
  </head>
  <body>
    <!-- body goes here -->
  </body>
</html>
```

结构、表现、行为三者分离

尽量在文档和模板中只包含结构性的 HTML；而将所有表现代码，移入样式表中；将所有动作行为，移入脚本之中。
在此之外，为使得它们之间的联系尽可能的小，在文档和模板中也尽量少地引入样式和脚本文件。
建议：

- 不使用超过一到两张样式表
- 不使用超过一到两个脚本（学会用合并脚本）
- 不使用行内样式 （`<style>.no-good {}</style>`）
- 不在元素上使用 style 属性 （`<hr style="border-top: 5px solid black">`）
- 不使用行内脚本 （`<script>alert('no good')</script>`）
- 不使用表象元素 （`i.e. <b>, <u>, <center>, <font>, <b>`）
- 不使用表象 class 名 （`i.e. red, left, center`）

使用 HTML5 的文档声明类型：`<!DOCTYPE html>`

- DOCTYPE标签是一种标准通用标记语言的文档类型声明，它的目的是要告诉标准通用标记语言解析器，它应该使用什么样的文档类型定义（DTD）来解析文档。
- 使用文档声明类型的作用是为了防止开启浏览器的怪异模式。
- 没有DOCTYPE文档类型声明会开启浏览器的怪异模式，浏览器会按照自己的解析方式渲染页面，在不同的浏览器下面会有不同的样式。
- 如果你的页面添加了<!DOCTYP>那么，那么就等同于开启了标准模式。浏览器会按照W3C标准解析渲染页面。

说到js和css的位置，大家应该都知道js放在下面，css放在上面。

> 所有浏览器中推荐:

```
<html>
  <head>
    <link rel="stylesheet" href="main.css">
  </head>
  <body>
    <!-- body goes here -->

    <script src="main.js" async></script>
  </body>
</html>
```

> 只兼容现代浏览器推荐:

```
<html>
  <head>
    <link rel="stylesheet" href="main.css">
    <script src="main.js" async></script>
  </head>
  <body>
    <!-- body goes here -->
  </body>
</html>
```

尽量在文档和模板中只包含结构性的 HTML；而将所有表现代码，移入样式表中；将所有动作行为，移入脚本之中。
在此之外，为使得它们之间的联系尽可能的小，在文档和模板中也尽量少地引入样式和脚本文件。
建议：

- 不使用超过一到两张样式表
- 不使用超过一到两个脚本（学会用合并脚本）
- 不使用行内样式 （`<style>.no-good {}</style>`）
- 不在元素上使用 style 属性 （`<hr style="border-top: 5px solid black">`）
- 不使用行内脚本 （`<script>alert('no good')</script>`）
- 不使用表象元素 （`i.e. <b>, <u>, <center>, <font>, <b>`）
- 不使用表象 class 名 （`i.e. red, left, center`）

严格模式

ECMAScript 5 严格模式可在整个脚本或独个方法内被激活。它对应不同的 javascript 语境会做更加严格的错误检查。严格模式也确保了 javascript 代码更加的健壮，运行的也更加快速。

严格模式会阻止使用在未来很可能被引入的预留关键字。

你应该在你的脚本中启用严格模式，最好是在独立的 IIFE 中应用它。避免在你的脚本第一行使用它而导致你的所有脚本都启动了严格模式，这有可能会引发一些第三方类库的问题。

变量声明

建议不再使用var，而使用let 和const 。优先使用const。

```
const [a,b,c] = [1,2,3];
```

- 静态字符串一律使用单引号或反引号，不建议使用双引号。动态字符使用反引号。

```
  const a = 'foobar';
  const b = foo${a}bar;
```

- 优先使用解构赋值

```
const arr = [1, 2, 3, 4];
const [first, second] = arr;
```

- 函数的参数如果是对象的成员，优先使用解构赋值。

```
function getFullName(obj) {
    const { firstName, lastName } = obj;
}

function getFullName({ firstName, lastName }) {
}
```

- 如果函数返回多个值，优先使用对象的解构赋值，而不是数组的解构赋值。这样便于以后添加返回值，以及更改返回值的顺序。

```
function processInput(input) {
    return { left, right, top, bottom };
}
const { left, right } = processInput(input);
```

使用严格等

总是使用 === 精确的比较操作符，避免在判断的过程中，由 JavaScript 的强制类型转换所造成的困扰。例如：

```
(function(log){
'use strict';


log('0' == 0); // true
log('' == false); // true
log('1' == true); // true
log(null == undefined); // true


var x = {
valueOf: function() {
return 'X';
}
};


log(x == 'X');


}(window.console.log));
```

等同== 和严格等===的区别

==， 两边值类型不同的时候，要先进行类型转换，再比较。

===，不做类型转换，类型不同的一定不等。

==等同操作符

如果两个值具有相同类型，会进行===比较，返回===的比较值

如果两个值不具有相同类型，也有可能返回true

如果一个值是null另一个值是undefined，返回true

如果一个值是string另个是number，会把string转换成number再进行比较

如果一个值是true，会把它转成1再比较，false会转成0

```
console.log( false == null ) // false
```

```
console.log( false == undefined ) // false
```

```
console.log( false == 0 ) // true
console.log( false == '' ) // true
console.log( false == NaN ) // false

console.log( null == undefined ) // true
console.log( null == 0 ) // false
console.log( null == '' ) // false
console.log( null == NaN ) // false

console.log( undefined == 0) // false
console.log( undefined == '') // false
console.log( undefined == NaN) // false

console.log( 0 == '' ) // true
console.log( 0 == NaN ) // false复制代码总结一下==
```

false 除了和自身比较为 true 外，和 0，"" 比较也为 true

null 只和 undefined 比较时为 true， 反过来 undefined 也仅和 null 比较为 true，没有第二个

0 除了和 false 比较为 true，还有空字符串 "" 和空数组 []

空字符串 '' 除了和 false 比较为 true，还有一个数字 0

==, >, <, +, -, ... 这些操作符所造成的隐式类型转换都是无副作用的，它不会改变变量本身保存的值。，但是，如果你覆写某个对象的 valueOf/toString的话，==就会产生副作用.

例如：

```
Array.prototype.valueOf = function() {
this[0]++;
return this;
}
var x = [1, 2, 3];
x == 0;
console.log(x); // [2, 2, 3]复制代码===操作符：
```

要是两个值类型不同，返回false

要是两个值都是number类型，并且数值相同，返回true

要是两个值都是stirng，并且两个值的String内容相同，返回true

要是两个值都是true或者都是false，返回true

要是两个值都是指向相同的Object，Arraya或者function，返回true

要是两个值都是null或者都是undefined，返回true

真假判断

js中以下内容为假：

- false
- null
- undefined
- 0
- '' (空字符串)
- NaN

三元条件判断（**if** 的快捷方法）

用三元操作符分配或返回语句。在比较简单的情况下使用，避免在复杂的情况下使用。没人愿意用 10 行三元操作符把自己的脑子绕晕。

不推荐：

```
if(x === 10) {
return 'valid';
} else {
return 'invalid';
}
```

推荐：

```
return x === 10 ? 'valid' : 'invalid'
```

ECMAScript 5 严格模式可在整个脚本或独个方法内被激活。它对应不同的 javascript 语境会做更加严格的错误检查。严格模式也确保了 javascript 代码更加的健壮，运行的也更加快速。

严格模式会阻止使用在未来很可能被引入的预留关键字。

你应该在你的脚本中启用严格模式，最好是在独立的 IIFE 中应用它。避免在你的脚本第一行使用它而导致你的所有脚本都启动了严格模式，这有可能会引发一些第三方类库的问题。

- 建议不再使用var，而使用let 和const 。优先使用const。

```
const [a,b,c] = [1,2,3];
```

- 静态字符串一律使用单引号或反引号，不建议使用双引号。动态字符使用反引号。

```
  const a = 'foobar';
  const b = foo${a}bar;
```

- 优先使用解构赋值

```
const arr = [1, 2, 3, 4];
const [first, second] = arr;
```

- 函数的参数如果是对象的成员，优先使用解构赋值。

```
function getFullName(obj) {
    const { firstName, lastName } = obj;
}

function getFullName({ firstName, lastName }) {
}
```

- 如果函数返回多个值，优先使用对象的解构赋值，而不是数组的解构赋值。这样便于以后添加返回值，以及更改返回值的顺序。

```
function processInput(input) {
    return { left, right, top, bottom };
}
const { left, right } = processInput(input);
```

总是使用 === 精确的比较操作符，避免在判断的过程中，由 JavaScript 的强制类型转换所造成的困扰。例如：

```
(function(log){
'use strict';


log('0' == 0); // true
log('' == false); // true
log('1' == true); // true
log(null == undefined); // true


var x = {
valueOf: function() {
return 'X';
}
};


log(x == 'X');


}(window.console.log));
```

等同== 和严格等===的区别

==， 两边值类型不同的时候，要先进行类型转换，再比较。

===，不做类型转换，类型不同的一定不等。

==等同操作符

如果两个值具有相同类型，会进行===比较，返回===的比较值

如果两个值不具有相同类型，也有可能返回true

如果一个值是null另一个值是undefined，返回true

如果一个值是string另个是number，会把string转换成number再进行比较

如果一个值是true，会把它转成1再比较，false会转成0

```
console.log( false == null ) // false
```

```
console.log( false == undefined ) // false
```

```
console.log( false == 0 ) // true
```

```
console.log( false == '' ) // true
```

```
console.log( false == NaN ) // false
```

```
console.log( null == undefined ) // true
console.log( null == 0 ) // false
console.log( null == '' ) // false
console.log( null == NaN ) // false

console.log( undefined == 0) // false
console.log( undefined == '') // false
console.log( undefined == NaN) // false

console.log( 0 == '' ) // true
console.log( 0 == NaN ) // false复制代码总结一下==
```

false 除了和自身比较为 true 外，和 0，"" 比较也为 true

null 只和 undefined 比较时为 true， 反过来 undefined 也仅和 null 比较为 true，没有第二个

0 除了和 false 比较为 true，还有空字符串 "" 和空数组 []

空字符串 '' 除了和 false 比较为 true，还有一个数字 0

==, >, <, +, -, ... 这些操作符所造成的隐式类型转换都是无副作用的，它不会改变变量本身保存的值。，但是，如果你覆写某个对象的 valueOf/toString的话，==就会产生副作用.

例如：

```
Array.prototype.valueOf = function() {
this[0]++;
return this;
}
var x = [1, 2, 3];
x == 0;
console.log(x); // [2, 2, 3]复制代码===操作符：
```

要是两个值类型不同，返回false

要是两个值都是number类型，并且数值相同，返回true

要是两个值都是stirng，并且两个值的String内容相同，返回true

要是两个值都是true或者都是false，返回true

要是两个值都是指向相同的Object，Arraya或者function，返回true

要是两个值都是null或者都是undefined，返回true

js中以下内容为假：

- false
- null
- undefined
- 0
- '' (空字符串)
- NaN

用三元操作符分配或返回语句。在比较简单的情况下使用，避免在复杂的情况下使用。没人愿意用 10 行三元操作符把自己的脑子绕晕。

不推荐：

```
if(x === 10) {
return 'valid';
} else {
return 'invalid';
}
```

推荐：

```
return x === 10 ? 'valid' : 'invalid'
```

id和class的命名

ID和class的名称总是使用可以反应元素目的和用途的名称，或其他通用的名称，代替表象和晦涩难懂的名称

> 不推荐：

```
.fw-800 {
font-weight: 800;
}
.red {
color: red;
}
```

> 推荐：

```
.heavy {
font-weight: 800;
}
.important {
color: red;
}
```

合理的使用ID

一般情况下ID不应该被用于样式，并且ID的权重很高，所以不使用ID解决样式的问题，而是使用class

> 不推荐：

```
#content .title {
font-size: 2em;
}
```

> 推荐：

```
.content .title {
font-size: 2em;
}
```

使用子选择器

很多前端开发人员写选择器链的时候不使用 直接子选择器（注：直接子选择器和后代选择器的区别）。

有时，这可能会导致疼痛的设计问题并且有时候可能会很耗性能。然而，在任何情况下，这是一个非常不好的做法。

如果你不写很通用的，需要匹配到DOM末端的选择器， 你应该总是考虑直接子选择器。

> 不推荐:

```
.content .title {
font-size: 2rem;
}
```

> 推荐

```
.content > .title {
font-size: 2rem;
}
```

尽量使用缩写属性

尽量使用缩写属性对于代码效率和可读性是很有用的，比如font属性。

> 不推荐：

```
border-top-style: none;
font-family: palatino, georgia, serif;
font-size: 100%;
line-height: 1.6;
padding-bottom: 2em;
padding-left: 1em;
padding-right: 1em;
padding-top: 0
```

> 推荐：

```
border-top: 0;
font: 100%/1.6 palatino, georgia, serif;
padding: 0 1em 2em;
```

0后面不带单位

省略0后面的单位。

> 不推荐：

```
padding-bottom: 0px;
margin: 0em;
```

> 推荐：

```
padding-bottom: 0;
margin: 0;
```

ID和class的名称总是使用可以反应元素目的和用途的名称，或其他通用的名称，代替表象和晦涩难懂的名称

> 不推荐：

```
.fw-800 {
font-weight: 800;
}
.red {
color: red;
}
```

> 推荐：

```
.heavy {
font-weight: 800;
}
.important {
color: red;
}
```

一般情况下ID不应该被用于样式，并且ID的权重很高，所以不使用ID解决样式的问题，而是使用class

> 不推荐：

```
#content .title {
font-size: 2em;
}
```

> 推荐：

```
.content .title {
font-size: 2em;
}
```

很多前端开发人员写选择器链的时候不使用 直接子选择器（注：直接子选择器和后代选择器的区别）。

有时，这可能会导致疼痛的设计问题并且有时候可能会很耗性能。然而，在任何情况下，这是一个非常不好的做法。

如果你不写很通用的，需要匹配到DOM末端的选择器， 你应该总是考虑直接子选择器。

> 不推荐:

```
.content .title {
font-size: 2rem;
}
```

> 推荐

```
.content > .title {
font-size: 2rem;
}
```

尽量使用缩写属性对于代码效率和可读性是很有用的，比如font属性。

> 不推荐：

```
border-top-style: none;
font-family: palatino, georgia, serif;
font-size: 100%;
line-height: 1.6;
padding-bottom: 2em;
padding-left: 1em;
padding-right: 1em;
padding-top: 0
```

> 推荐：

```
border-top: 0;
font: 100%/1.6 palatino, georgia, serif;
padding: 0 1em 2em;
```

省略0后面的单位。

> 不推荐：

```
padding-bottom: 0px;
margin: 0em;
```

> 推荐：

```
padding-bottom: 0;
margin: 0;
```

前端采用git代码管理工具

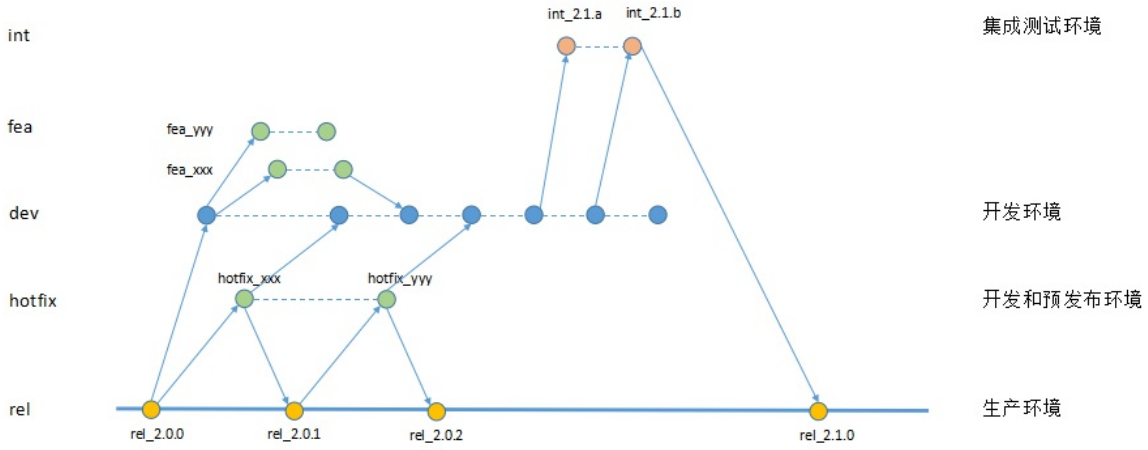
线上地址http://git.ydxcx.net

pc端商城官网后台代码地址：git@git.ydxcx.net:ys/weishier-admin-core.git

官网小程序代码地址：git@git.ydxcx.net:ys/websitesmallProgram.git

商城小程序代码地址：git@git.ydxcx.net:ys/MallSP.git

代码分支管理



hotfix_xxx: 其中xxx是表示从rel分支哪个版本拉出来；

fea_xxx: 其中xxx用时间表示：例如：201808281340

方法：ajaxGetLogin

说明：以get方式提交ajax信息（需要验证登录信息未登录跳转到登录页）

参数：opt 类型：object，对象具体参数如下：

参数名	是否必填	默认	类型	备注
type	否	'GET'	string	
dataType	否		string	预期服务器返回的数据类型
contentType	否	"application/x-www-form-urlencoded"	string	发送信息至服务器时内容编码类型。
data	否		string	ajax传参(data和SDatad都有data优先)
SData	否		Object	ajax传参（转为json字符串）
url	是		string	调用接口地址
success	否		function	成功回调方法(success和_success都有success优先)
_success	否		function	成功回调方法(默认增加了成功是否提示)
error	否		function	成功回调方法(error和_error都有error优先)
_error	否		function	成功回调方法(默认增加了提示)
msg	否		string	_success成功提示的信息，不填写不会提示信息

方法：ajaxPostLogin

说明：以post方式提交ajax信息（需要验证登录信息未登录跳转到登录页）

参数：opt 类型：object，对象具体参数如下：

参数名	是否必填	默认	类型	备注
type	否	'POST'	string	
dataType	否	'json'	string	预期服务器返回的数据类型

contentType	否	'application/json;charset=UTF-8'	string	发送信息至服务器时内容编码类型。
data	否		string	ajax传参(data和SDatad都有data优先)
SData	否		Object	ajax传参（转为json字符串）
url	是		string	调用接口地址
success	否		function	成功回调方法(success和_success都有success优先)
_success	否		function	成功回调方法(默认增加了成功是否提示)
error	否		function	成功回调方法(error和_error都有error优先)
_error	否		function	成功回调方法(默认增加了提示)
msg	否		string	_success成功提示的信息，不填写不会提示信息

方法名：**ajaxPost**

说明：以post方式提交ajax信息

参数：opt 类型：object，对象具体参数如下：

参数名	是否必填	默认	类型	备注
type	否	'POST'	string	
dataType	否	'json'	string	预期服务器返回的数据类型
contentType	否	'application/json;charset=UTF-8'	string	发送信息至服务器时内容编码类型。
data	否		string	ajax传参(data和SDatad都有data优先)
SData	否		Object	ajax传参（转为json字符串）
url	是		string	调用接口地址

success	否		function	成功回调方法(success和_success都有success优先)
_success	否		function	成功回调方法(默认增加了成功是否提示)
error	否		function	成功回调方法(error和_error都有error优先)
_error	否		function	成功回调方法(默认增加了提示)
msg	否		string	_success成功提示的信息，不填写不会提示信息

方法：ajax

说明：提交ajax信息

参数：opt 类型：object，对象具体参数如下：

参数名	是否必填	默认	类型	备注
type	否	'GET'	string	
dataType	否		string	预期服务器返回的数据类型
contentType	否	"application/x-www-form-urlencoded"	string	发送信息至服务器时内容编码类型。
data	否		string	ajax传参(data和SDatad都有data优先)
SData	否		Object	ajax传参（转为json字符串）
url	是		string	调用接口地址
success	否		function	成功回调方法(success和_success都有success优先)
_success	否		function	成功回调方法(默认增加了成功是否提示)
error	否		function	成功回调方法(error和_error都有error优先)
_error	否		function	成功回调方法(默认增加了提示)

msg	否		string	_success成功提示的信息，不填写不会提示信息

页面引用：

```
    <script src="../../../v12_js/components/mall/goodslayer.js"></script>
    <script src="../../../v12_js/components/mall/goodsLayerModel.js"></script>
```

js组件引用：

```
new Vue({
    components:{
         'goodslayer': goodsLayerModel,//商品选择弹窗
    }
});
```

调用js配置

```
goodslayerConfig:{
            isshowhead:true,//是否显示头部
            selectType:"goods",//默认选择的头部类型category:商品分类、goods:商品列表
            storeId:0, //门店id,默认取的是localStorage.getItem('decStoreId')
            goodsconfig:{//商品列表
                 head: {
                    isshow: true,//是否显示
                    name: "商品列表"//显示头部名称
                 }, //头部
                 search: {
                    isshow: true,//是否显示
                    name: ""//搜索默认名称
                }, //搜索
                page: {
                    isshow: true,//是否显示
                    currentpage: 1, //当前页
                    totalPage: 1, //总页数
                    totalSize: 0,
                    pagesize: 5,
                },
                showType: 1, //1：单选显示列表，2：显示树形结构,3：多选列表
            },
            categoryconfig:{//商品分类
                head: {
                    isshow: true,//是否显示
                    name: "商品分类"//显示头部名称
                }, //头部
            }
            }
```

html页面调用

```
<goodslayer v-model="goodslayerConfig.isShow" :mask-closable="false" width="560"
                   title="选择拼团商品" @on-ok="saveLinks" :items="postdata" :config="go
odslayerConfig"></goodslayer>
```

回调取值

```
/**
         * @description 选择链接地址-保存
         * */
        saveLinks: function (result) {
            var that = this;

            //如果result.ids的长度为0，则提示选择
            if (!result.ids) {
                that.$Modal.warning({
                    title: '提示',
                    content: '请选择商品！'
                });

                return false;
            } else {

                //result.ids为所选商品
            }
        },
```

页面引用：

```
    <script src="../../../v12_libs/vue_crooper/vue-crooper0.34.js" type="text/javascrip
t" charset="utf-8"></script>
    <script src="../../../v12_js/components/cropperLayer.js" type="text/javascript" cha
rset="utf-8"></script>
```

html页面调用

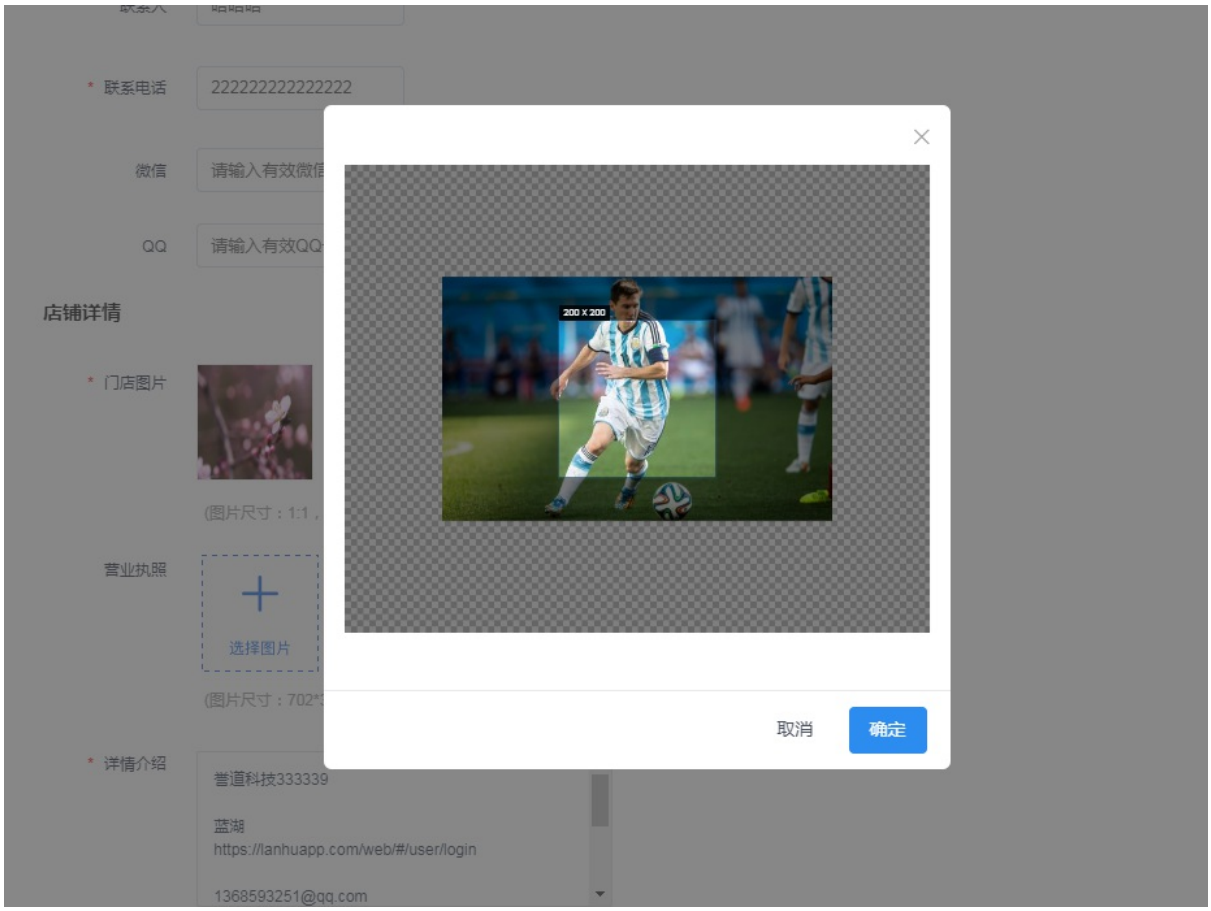
```
<input id="fileToUpload_1" class="upload_input" name="fileUpload" type="file" accept="i
mage/png, image/jpeg, image/jpg" tabindex=-1 @change="uploadImg($event, [200, 200])" />

<!-- 图片裁剪组件 beginning -->
<cropper-layer ref="cropperFn" @on-ok="finish"></cropper-layer>
<!-- 图片裁剪组件 end -->
```

js组件引用：

```
new Vue({
      el: '#app',
      data: {

      },
      components: {
        'cropper-layer': cropperLayer,  //图片裁剪弹层
      },
      methods: {
          /**
           * @description 点击上传图片按钮
           * @param e {Object} event对象
           * @param cropArr {Array} 图片像素[长, 宽]，eg. [702, 360]
           * */
          uploadImg: function(e, cropArr){
              this.$refs.cropperFn.uploadImg(e, cropArr);
          },
          /**
           * @description 点击弹层确定按钮，上传图片
           * @param res {Object} 上传图片接口，回调数据
           * */
          finish: function(res){
              console.log(res);
          }
      },

      mounted: function () {

      }
```

```
})
```

组件截图


联系电话
222222222222222
微信
请输入有效微信
QQ
请输入有效QQ
店铺详情
门店图片
(图片尺寸：1:1，
营业执照
选择图片
(图片尺寸：702*
详情介绍
誉道科技333339
蓝湖
https://lanhuapp.com/web/#/user/login
1368593251@qq.com
200 x 200
取消
确定

页面引用：

```
    <link rel="stylesheet" type="text/css" href="../../../v12_css/components/public.css
?v=0.001" />
    <script src="../../../v12_js/components/Upload.js"></script>
    <script src="../../../v12_js/components/fileUpload.js"></script>
    <script src="../../../v12_js/components/imageUpload.js"></script>
    <script src="../../../v12_libs/vue_crooper/vue-crooper0.34.js" type="text/javascrip
t" charset="utf-8"></script>
```

js组件引用：

```
new Vue({
    components:{
         'imageupload': imageUpload,//上传图片
    }
});
```

```
     //上传类型 image，video
        type: {
            type: String,
            default: "image"
        },
        //上传最大数量
        uploadMaxNum: {
            type: Number,
            default: 1
        },
        //备注[format]取文件格式数据
        Tip: {
            type: String,
            default: "（注：仅支持上传[format]格式，建议上传尺寸：[SWidth]*[SHeight]px。）
"
        },
        //是否一开始启动裁剪
        isCropper: {
            type: Boolean,
            default:false
        },
        //是否显示删除按钮
        isShowDel: {
            type: Boolean,
            default: true
        },
        //上传按钮文本
        btnText: {
```

```
        type: String,
        default: "上传图片"
    },
    //进度条长度
    PWidth: {
        type: Number,
        default: 640
    },
    //浏览图片宽度
    SWidth: {
        type: Number,
        default: 640
    },
    //浏览图片高度
    SHeight: {
        type: Number,
        default: 480
    },
    //文件
    files: {
        type: [String, Array],
    },
    //进度条宽度
    strokeWidth: {
        type: Number,
        default: 2
    },
    //文件格式
    format: {
        type: Array,
        default() {
            return [];
        }
    },
    //最大大小
    maxSize: {
        type: Number,
        default: 1024,
    },
    //格式错误信息
    formatError: {
        type: String,
        default: function () {
            if (this.format.length > 1) {
                return '请上传[format]格式的图片!'
            } else {
                return '请上传正确的图片格式'
            }
        }
```

```
        },
        //最大大小错误信息
        maxSizeError: {
            type: String,
            default: function () {
                return '请上传小于[maxSize]kb的图片文件！'
            }
        },
        //成功回调函数
        onSuccess: {
            type: Function,
        },
        //删除回调函数
        onRemove: {
            type: Function,
        },
        //上传地址
        uploadurl: {
            type: String,
        },
        //是否显示默认文件
        isShowDefault: {
            type: Boolean,
            default: false
        },
        //默认图片
        DefaultFile: {
            type: String,
            default: 'http://www.ydxcx.net/group1/M00/00/AA/rBEACVrQTs2AKLcQAAAUXfgp-rg
143.png'
        },
```

html页面调用

```
<image-upload :onSuccess="imagesuccess"   :s-width="250" :s-height="100" :files="item.p
icturePath"  :onRemove="imageRemove" ></image-upload>
```

回调取值

```
imagesuccess: function (data, file, index,res) {
            if (data.code == 0) {
                this.componentData.contentList[this.currentIndex].picturePath = data.ur
l;
            } else {
                api.showToast("fail", "上传失败");
            }
        },
```

```
        imageRemove: function (index) {
            this.componentData.contentList[this.currentIndex].picturePath = "";
        },
```

页面引用：

```
    <link rel="stylesheet" type="text/css" href="../../../v12_css/components/public.css
?v=0.001" />
    <script src="../../../v12_js/components/Upload.js"></script>
    <script src="../../../v12_js/components/fileUpload.js"></script>
    <script src="../../../v12_js/components/videoUpload.js"></script>
```

js组件引用：

```
new Vue({
    components:{
         'videoupload': videoUpload,//上传视频
    }
});
```

```
    //最大大小
        maxSize: {
            type: Number,
            default: 100240,
        },
        //上传最大数量
        uploadMaxNum: {
            type: Number,
            default: 1
        },
        //备注
        Tip: {
            type: String,
            default: "（注：仅支持上传[format]格式，建议上传文件大小[maxSize]以内,视频尺寸1
6:9。）"
        }
        //上传按钮文本
        btnText: {
            type: String,
            default: "上传视频"
        },
        //进度条长度
        PWidth: {
            type: Number,
            default: 640
        },
        //浏览图片宽度
        SWidth: {
            type: Number,
            default: 640
```

```
        },
        //浏览图片高度
        SHeight: {
            type: Number,
            default: 480
        },
        //文件
        files: {
            type: [String, Array],
        },
        //进度条宽度
        strokeWidth: {
            type: Number,
            default: 2
        },
        //文件格式
        format: {
            type: Array,
            default() {
                return ['mp4'];
            }
        },
        //成功回调函数
        onSuccess: {
            type: Function,
        },
        //删除回调函数
        onRemove: {
            type: Function,
        },
        //格式错误信息
        formatError: {
            type: String,
            default: function () {
                if (this.format.length > 1) {
                    return '请上传[format]格式的视频!'
                } else {
                    return '请上传正确的视频格式'
                }
            }
        },
        //最大大小错误信息
        maxSizeError: {
            type: String,
            default: function () {
                return '请上传小于[maxSize]的视频文件！'
            }
        },
```

```
        //是否显示默认文件
        isShowDefault: {
            type: Boolean,
            default: true
        },
        //默认图片
        DefaultFile: {
            type: String,
            default: ''
        },
```

html页面调用

```
<videoUpload :onSuccess="videosuccess" :files="componentData.videoUrl" :onRemove="video
Remove" ></videoUpload>
                            </div>
```

回调取值

```
videosuccess: function (data) {
            if (data.code == 0) {
                this.componentData.videoUrl = data.data.videoUrl;//给图片参数picturePath
赋值
                if (!this.componentData.videoLogo) {
                    this.componentData.videoLogo = data.data.picUrl;//给图片参数pictureP
ath赋值
                }
            } else {
                api.showToast("fail", "上传失败");
            }
        },
        videoRemove: function () {
            this.componentData.videoUrl = "";
        },
```