**前端代码规范**

**Author:fangfang.Peng**

**Date: 2018-09-05**

1. **前端普适性规范**

不管有多少人共同参与同一项目，一定要确保每一行代码都像是同一个人编写的。

1.1项目命名：

a.项目命名全部采用小写方式，以中划线分隔，禁止驼峰式命名。如：my-project-name.

1.2文件命名:

●React组件的文件命名统一使用index.jsx，其中的类名使用驼峰式且首字母大写，类名与当前文件上级目录名称对应,如：component-test/index.jsx，类名为ComponentTest.

●html文件命名采用中划线形式,如：error-report.html

●文件名中只可由小写英文字母a~z、排序数字0~9或间隔符-组成，禁止包含特殊符号，比如空格、$等.

●文件名的命名要见名知意，名称过长是可以考虑适当使用缩写形式。缩写只用来表示结构，如col、nav、btn等，不可自造缩写。

●不允许命名带有广告等英文的关键词，例如：ad、adv、adver、advertising等，防止该模块被浏览器当做垃圾广告过滤掉。任何文件的命名均如此。

1. **HTML规范**

2.1使用四个空格的缩进，保证代码在各种环境下显示一致的唯一方式。

2.2嵌套的节点应该缩进一个tab（4个空格）。

2.3在属性值固定的属性上，使用双引号，不要使用单引号。

2.4不要忽略关闭标签。

2.5减少标签数量。尽可能的避免多余的父节点，可以参考如下：

<!—not so good -->

<span class=”avatar”>

<img src=”…”/>

</span>

<!— good -->

<img src=”…”/>

1. **CSS规范**

3.1使用四个空格的缩进；

3.2为了代码的易读性，在每个声明的左括号前增加一个空格:

//not so good

span{

font-size: 14px;

}

// good

span {

font-size: 14px;

}

3.3声明块的右括号应该另起一行：

// not so good

span {

font-size: 14px;

}

.title {

font-weight: bold;

}

// good

span {

font-size: 14px;

}

// 此处是空行

.title {

font-weight: bold;

}

3.4每条样式声明:后应该插入一个空格：

// not so good

span {

font-size:14px;

}

// good

span {

font-size: 14px;

}

3.5每条声明应该只占用一行：

// not so good

span {

font-size: 14px; color: #3089dc;

}

// good

span {

font-size: 14px;

color: #3089dc;

}

3.6所有声明应该加以分号结尾，虽然最后一条声明后的分号是可选的，但是如果没有，你的代码会更容易出错：

// not so good

span {

font-size: 14px;

color: #333333

}

// good

span {

font-size: 14px;

color: #333333;

}

3.7逗号分隔的属性，都应该在逗号之后增加一个空格:

// not so good

span {

font-family: Arial,Verdana,Geneva,sans-serif;

}

// good

span {

font-family: Arial, Verdana, Geneva, sans-serif;

}

3.8不要在颜色值rgb()、rgba()、hsl()、hsla()、和rect()中增加空格,并且不要带有取值前面不必要的0（如：使用.5代替0.5）:

// not so good

span {

background: rgba(0, 0, 0, 0.5);

}

// good

span {

background: rgba(0,0,0,.5);

}

3.9尽可能的使用短的十六进制数值,如#aabbcc可以简写成#abc：

// not so good

span {

background: #cccccc;

}

// good

span {

background: #ccc;

}

3.10不要为0指明单位，比如使用margin: 0,而不是margin: 0px;

3.11相关属性的声明应该以如下顺序分组处理：

●Positioning

●Box model盒模型

●Typographic排版

●Visual外观

Positioning处在第一位，因为它可以使一个元素是否脱离正常文本流，并且覆盖盒模型相关的样式。盒模型紧跟其后，因为它决定了一个组件的大小和位置。其他属性只在组件内部起作用或者不会对前面两种情况的结果产生影响，所以排在后面。另外一方面，这个浏览器渲染元素的性能也有一定的关系：

.decoration-order {

/\* Positioninig \*/

position: absolute;

top: 0;

right: 0;

bottom: 0;

left: 0;

z-index- 100;

/\* Box-model \*/

display: block;

float: right;

width: 100px

height: 100px;

/\* Typography \*/

font: sans-serif;

line-height: 1.5;

color: #333;

text-align: center;

/\* Visual \*/

background-color: #ccc;

border: 1px solid #e5e5e5;

border-radius: 3px;

/\* Misc \*/

opacity: 1;

}

3.12媒体查询：

尽量将媒体查询的位置靠近他们相关的规则,或者统一放在文档底部。若统一放在文档底部的时候，顺序应按照正常样式进行布局且注释统一。

3.13前缀属性：

当使用厂商前缀属性时，通过缩进使取值垂直对齐以便进行多行编辑：

// good

.selector {

-webkit-border-radius: 3px;

-moz-border-radius: 3px;

-ms-border-radius: 3px;

-o-border-radius: 3px

border-radius: 3px;

}

3.14单条声明的声明块：

在一个声明块中只包含一条声明的情况下，为了易读性和快速编辑，可以考虑移除其中的换行。所有包含多条声明的声明块应该分为多行。

// good

.span { width: 60px; }

.span { width: 160px; }

.span { width: 260px; }

3.15属性简写：

尽量不使用属性简写的方式，属性简写必须显示设置所有的取值。常见的可简写属性有：padding, margin, font, background, border, border-radius等。

大多数情况下，其实并不需要设置属性简写中包含的所有值。如只设置元素上下margin,所以如果需要，只设置这两个值。过度使用简写往往会导致更混乱的代码，其中可能包含不必要的重写和意想不到的副作用，例如：

// not so good

.element {

margin: 0 0 10px;

background: red;

background: url(“image.png”);

border-radius: 3px 3px 0 0;

}

// good

.element {

margin-bottom: 10px;

background-color: red;

background-image: url(“image.png”);

border-top-left-radius: 3px;

border-top-right-radius: 3px;

}

3.16Less和Sass中的嵌套

避免不必要的嵌套。如果需要嵌套，嵌套深度最好不要超过三层。

3.17选择器：

如无必要，不得为id、class选择器添加类型选择器进行限定，在性能和维护性上都有一定的影响。

// not so good

div#error, p.message {

font-size: 14px;

}

// good

#error, . message {

font-size: 14px;

}

3.18 a标签伪类书写顺序：

<a>标签的四个伪类的特殊性是一样的，当涉及伪类书写时，书写顺序很重要，应该书写的顺序如下（LVHA, 形象的可以这样记住：我买了一个LV， 哈哈(HAHA)）：

link -> visited -> hover -> active;

3.19当一个rule包含了多个selector时，每个选择器声明必须独占一行

// good

.post,

.page,

.commit {

line-height: 1.5;

}

// bad

.post, .page, .commit {

line-height: 1.5;

}

3.20 > + ~ 选择器的两边各保留一个空格

// good

div > span {

padding: 10px;

}

// bad

div>span {

padding: 10px;

}

3.21属性选择器中的值必须使用双引号包围

// good

div[character=”juliet”] {

font-size: 14px;

}

// bad

div[character=’juliet’] {

font-size: 14px;

}

3.22属性定义，必须另起一行

// good

.selector {

padding: 0;

margin: 0;

}

// bad

.selector {

padding: 0; margin: 0;

}

3.23 !important的使用

建议尽量不要使用!important声明。

当需要强制指定样式且不允许任何场景覆盖时，通过标签内联和!important定义样式。需要注意的是，仅仅在设计上确实不允许任何其他场景覆盖样式时，才是用内联的!important样式。

3.24 url。

url()函数中的路径不加引号。

url()函数中的绝对路径可省去协议名。

body {

background: url(//baidu.com/img/bg.png) no-repeat 0 0;

}

3.25 16进制颜色值中的英文字符统一使用小写。

3.26 font-weight属性使用数值方式描述。

CSS的字重分100~900共九档，但目前受字体本身和浏览器的限制，实际上支持400和700两档，分别对应关键词normal和bold。

3.27 line-height在定义文本段落时，应使用数值。

将 line-height 设置为数值，浏览器会基于当前元素设置的 font-size 进行再次计算。在不同字号的文本段落组合中，能达到较为舒适的行间间隔效果，避免在每个设置了 font-size 都需要设置 line-height。

3.28【强制】使用transition时应制定transition-property。

// good

.box {

transition: color 1s, border-color 1s;

}

// bad

.box {

transition: all 1s;

}

3.29尽可能在浏览器能高效实现的属性上添加过渡和动画.

在可能的情况下应选择这样四种变换:

transform: translate(npx, npx);

transform: scale(n);

transform: rotate(ndeg);

transform: 0..1;

// good

.box {

transition: transform 1s;

}

.box: hover {

transform: translate(20px);

}

// bad

.box {

left: 0;

transition: left 1s;

}

.box: hover {

left: 20px;

}

1. **JS规范**

4.1变量命名：

命名变量时，不仅要遵循变量命名有效字符规范，命名的语义化也非常重要。另外，表达式中的运算符与操作数之间需要空格：

// bad

let str = ‘john’;

let flag=true;

// good

let firstName = ‘john’;

let isActive = true;

常量名称使用全大写：

// bad

const pi = 3.141592654;

// good

const PI = 3.141592654;

使用驼峰式写法：

// bad

const displayname = ‘alidw’;

// good

const displayName = ‘alidw’;

附：常见的几种命名法说明

●camel驼峰式(小驼峰)命名法。形如thisIsAnApple

●pascal 帕斯卡(大驼峰)命名法。形如ThisIsAnApple

●下划线命名法。形如this\_is\_an\_apple

●中划线命名法。形如this-is-an-apple

a).变量名：必须使用camel命名法

b).参数名：必须使用camel命名法

c).函数名：必须使用camel命名法

d).方法/属性：必须使用camel命名法

e).常量名：必须使用全部大写的下划线命名法，如IS\_DEBUG\_ENABLED

f).类名：必须使用pascal命名法

g).枚举名：必须使用pascal命名法

h).枚举的属性：必须使用全部大写的下划线命名法

i).命名空间：必须使用camel命名法

4.2 总是使用分号：

如果仅仅依靠语句间的隐式分隔，有时会很麻烦。很明显，你自己更清楚哪里是语句的起止，哪里该加分号，就加上。

// bad

const obj = {name: ‘nkis’}

// good

const obj = {name: ‘nkis’};

4.3代码格式化：

●数组和对象的初始化，如果初始值不是很长，就保持写在单行上：

const arr = [1, 2, 3];

const obj = {a: 1, b: 2, c: 3};

●对象最后一个属性值之后不加逗号

// bad

let obj = {

name: ‘alidw’,

age: 7,

}

// good

let obj = {

name: ‘alidw’,

age: 7

}

4.4引号的使用

使用单引号定义字符串，当字符串中包含单引号的时候，可适当使用双引号定义字符串。适当使用ES6的模板字符串。

// not so good

let displayName = “alidw”;

// good

let displayName = ‘alidw’;

4.5函数、方法的写法

●函数声明式，函数名后紧跟括号，右括号与大括号之间增加一个空格，多个参数之间需在逗号后面加空格：

// not so good

function func (name,age){

console.log(name,age);

}

// good

function func(name, age) {

console.log(name, age);

}

●函数表达式

函数表达式，需要在函数最后的大括号之后增加结尾分号:

// not so good

const func = function() {}

// good

const func = function() {};

ES6箭头函数，也属于函数表达式，也需要在函数结尾添加分号。

const func = () => {console.log(‘100’)};

4.6对象属性的获取

推荐使用ES6的解构赋值语法。

// not so good

let name = obj.name;

let age = obj.age;

// good

const { name = ‘defaultName’, age = 18} = obj;

注意上面的写法，ES6解构支持赋值默认值，当且仅当对象解构的属性值严格等于undefined时，默认值才会赋值生效；

4.7数组、对象的创建

必须使用数组和对象字面量来代替数组和对象构造器。

// bad

const a1 = new Array(3);

const b1 = new Object();

// good

const a1 = [1, 2, 3];

const b1 = {name: ‘tom’, age: 20};

4.8循环的使用

在循环中，循环变量一次获取，多次使用。若放在循环中每次进行判断，会影响性能。虽然现代浏览器的性能已经做得很好，但这不应该是滥用循环变量的使用方式的借口。

// bad

for (let i = 0; i < arr.length; i++) {}

// good

for (let i = 0, len = arr.length; i < len; i++) {}

另外，js数值还提供一系列函数可供使用，如forEach、map、every、some、reduce等也可以循环处理数组。

4.9注释

●函数注释：

/\*\*

\* 简述

\* @param <string> arg1 参数1

\* @param <boolean> arg2 参数2

\*/

function func(arg1, arg2) {}

●单行注释:

单独一行：//与注释文字之间保留一个空格；

// not so good

//这里调用了数组去重方法，去除数组重复值

duplicateRemoval(arr);

// good

// 这里调用了数组去重方法，去除数组重复值

duplicateRemoval(arr);

●多行注释：

多行注释如下:

/\* const arr = [1, 2,3 ];

const name = ‘’; \*/

●方法中建议添加必要的代码注释。

4.10关键字后面加空格

// not so good

if(condition) { … }

// good

if (condition) { … }

4.11字符串拼接操作符之间要加空格

// not so good

const message = ‘hello’+’world’;

// good

const message = ‘hello’ + ’world’;

4.12 else关键字要与花括号保持在同一行

// not so good

if (condition) {

}

else {

}

// good

if (condition) {

} else {

}

4.13不允许有连续多行空行

// bad

let value = ‘hello world’;

console.log(value);

// good

let value = ‘hello world’;

console.log(value);

4.14三元运算符的使用

对于三元运算符，?和：与他们所负责的代码处于同一行。

// good

let location = env.devepment ? ‘localhost’ : ‘www.api.com’;

let location = env.develpment

? ‘localhost’

: ‘www.api.com’;

// bad

let location = env.develpment ?

‘localhost’ :

‘www.api.com’;

4.15单行代码块两边加空格

// good

function func () { return true; }

// not so good

function func () {return true;}

4.16点号操作符必须与属性在同一行

// avoid

console

.log(‘100’);

// good

console.log(100);

4.17文件末尾留一空行，换行。

4.18函数调用时标识符与括号之间不留间隔

// not so good

console.log (100);

// good

console.log(100);

4.19键值对当中冒号与值之间要留空白

// avoid

let obj = { key : ‘value’ };

let obj = { key :‘value’ };

let obj = { key:‘value’ };

// good

let obj = { key: ‘value’ };

4.20构造函数名要以大写字母开头

// avoid

function animal () {}

let dog = new animal();

// good

function Animal () {}

let dog = new Animal();

另外，无参的构造函数调用时要带上括号：let pig = new Animal; 这种写法要避免。

4.21对象中定义了存值器，必须要有对应的取值器

// avoid

const person = {

set name (value) {

this.name = value;

}

};

// good

const person = {

set name (value) {

this.name = value;

},

get name () {

return this.name;

}

};

4.22子类的构造器中一定要调用super,如果不调用父类构造函数，就无法正确获取this。

class Dog extends Animal {

constructor () {

super();

}

}

4.23代码中不要使用debugger

避免代码中出现的debugger!

4.24不要扩展原生对象

// avoid

Object.prototype.age = 21;

4.25避免不必要的布尔转换

// avoid

const result = true;

if (!!result) {

// …

}

4.26 switch语句中，除非直接返回，否则一定要使用break来将条件分支正常中断。

4.27使用getPrototypeof来代替\_\_proto\_\_。

4.28函数的返回值不能与return成两行分开。

// avoid

function func() {

return

{

name: ‘alidw’

}

}

// good

function func() {

return {

name: ‘alidw’

}

}

4.29使用this前确保super()已调用

// avoid

class Dog extends Animal {

constructor () {

this.legs = 4;

super();

}

}

4.30如果有更好的实现，尽量不要使用三元运算符

// avoid

let score = val ? val : 0;

// good

let score = val || 0;

4.31禁止使用with

// avoid

with (val) { … }

4.31遇到分号时空格要后留，不前留

// bad

for (let i = 0 ;i < items.length ;i++) { … }

// good

for (let i = 0; i < items.length; i++) { … }

4.32圆括号间不留空格

// not so good

getName( name );

// good

getName(name);

4.33模板字符串中变量前后不加空格

// not so good

const message = `hello, ${ name }`;

// good

const message = `hello, ${name}`;

4.34判断变量数据类型推荐使用toString方法

Object.prototype.toString.call(true) // “[object Boolean]”

Object.prototype.toString.call(100) // “[object Number]”

Object.prototype.toString.call(‘str’) // “[object String]”

Object.prototype.toString.call({}) // “[object Object]”

Object.prototype.toString.call([]) // “[object Array]”

Object.prototype.toString.call(null) // “[object Null]”

Object.prototype.toString.call(undefined) // “[object Undefined]”

Object.prototype.toString.call(new RegExp(‘’)) // “[object RegExp]”

4.35函数形参个数大于等于4个，封装对象传参。

// not so good

function func(name, age, sex, weight, school) {

console.log(name, age, sex, weight, school);

}

func(‘liming’, 28, ‘男’, 80, ‘清华大学’);

// better

function func(obj) {

const { name, age, sex, weight, school } = obj;

console.log(name, age, sex, weight, school);

}

func({ name: ‘liming’, age: 28, sex: ‘男’, weight: 80, school: ‘清华大学’’ });

4.36类中的每个方法之间使用空行分隔。

// not so good

class Student {

getName () {

}

getAge() {

}

}

// good

class Student {

getName () {

}

//此处是空行

getAge() {

}

}

4.37过长的逻辑条件组合

当因为较复杂的逻辑条件组合导致80个字符无法满足需求时，应当将每个条件独立一行，逻辑运算符放置在行首进行分隔，或将部分逻辑按逻辑组合进行分隔。最终将右括号)与左大括号{放在独立一行，保证与if内语句块能容易视觉辨识。

如：

if (user.isAuthenticated()

&& user.isWake

&& user.hasAuthority(‘add-admin’

|| flag

) {

}

4.38在进行对象结构的过程中，如果解构的属性较多，添加换行显示：

// good

const {

activeInfo,

index,

dateTime,

backGround

} = obj;

当一行解构显示不下时，增加换行。

**五、React组件相关代码规范**

5.1原则上，每个通用组件的jsx文件和样式文件放在同一层目录下。

--|DatePicker

----|index.jsx

----|index.scss

5.2组件代码缩进。

代码缩进遵循以下格式：

// bad

<Foo superLongParam=”bar”

anotherSuperLongParam=”baz” />

// good 如果有多行属性的话，新建一行关闭标签

<Foo superLongParam=”bar”

anotherSuperLongParam=”baz”

/>

5.3 render() 方法始终放在最后，其他顺序如下：

●displayName 属性名

●statics

●propTypes

●constructor

#声明周期函数开始

●componentWillMount() {}

●componentDidMount() {}

●componentWillReceiveProps() {}

●shouldComponentUpdate() {}

●componentWillUpdate() {}

●componentDidUpdate() {}

●componentWillUnmount() {}

●handleClick

●render() {}

5.4将多行JSX标签使用括号括起来，单行可以不需要。

// good

getContent() {

const result = (

<div>

<span>标签</span>

</div>;

);

}

// good

getContent() {

const result = <div>标签</div>;

return result;

}

// not so good

getContent() {

const result = <div>

<span>标签</span>

</div>;

}

5.5 JSX固定属性值总是使用双引号，其他均使用单引号。

// good

<Foo bar=”bar” />;

<Foo style={{ font-size: ‘14px’ }} />

// not so good

<Foo bar=’bar’ />;

<Foo style={{ font-size: “14px” }} />

5.6类属性和类方法之间添加空行分隔。

// good

class Student {

name = ‘lilei’;

age = 20;

getName() {

}

}

// not so good

class Student {

name = ‘lilei’;

age = 20;

getName() {

}

}

5.7事件回调函数使用箭头函数，保证上下文的正确获取。

5.8类方法中，箭头函数最后要加分号；函数声明式方法不加分号。

5.9使用ES class 声明组件，禁止使用React.createClass

React v15.5.0已经弃用了React.createClass函数。

// bad

let message = React.createClass({

render() {

return <span>{this.state.message}</span>;

}

});

// good

class Message extends PureComponent {

render() {

return <span>{this.state.message}</span>;

}

}

5.10所有组件均需声明propTypes和defaultProps。

5.11对于所有非isRequired的属性，在defaultProps中声明对应的值。声明初始值有助于对组件初始状态的理解，也可以减少propTypes对类型进行校验产生的开销。

5.12没有子节点的非DOM组件使用自闭合语法。

对于DOM节点，按照HTML编码规范相关规则进行闭合。

// bad

<Foo></Foo>

// good

<Foo />

5.13保持起始和结束标签在同一层缩进。

// bad

class Message {

render() {

return <div>

<span>hello world</span>

</div>;

}

}

// good

class Message {

render() {

return (

<div>

<span>hello world</spam>

</div>

);

}

}

5.14对于直接return的函数组件，可以直接是用括号，省去大括号和return关键字。

let Message = () => (

<div>

<span>hello world</spam>

</div>

);

5.15对于多属性组件，需要换行。从第一个属性开始，每一个属性一行。

// good

<SomeComponent

longProp={longProp}

anotherLongProp={anotherLongProp}

/>

// good

<SomeComponent

longProp={longProp}

anotherLongProp={anotherLongProp}

>

<SomeChild />

<SomeChild />

</ SomeComponent>

5.16以字符串字面量作为值的属性使用双引号（"），在其它类型表达式中的字符串使用单引号（'）。

// bad

<Foo bar='bar' />

<Foo style={{ width: "20px" }} />

// good

<Foo bar="bar" />

<Foo style={{ width: '20px' }} />

5.17自闭合标签的/>前添加一个空格。

// bad

<Foo bar="bar"/>

<Foo bar="bar" />

// good

<Foo bar="bar" />

5.18对于值为true的属性，可以省去值的部分。

// bad

<Foo visible={true} />

// good

<Foo visible />

5.19对于需要使用key的场合，提供一个唯一标识作为key属性的值，禁止使用可能会变化的属性（如索引）。key属性是React在进行列表更新时的重要属性，如该属性会发生变化，渲染的性能和正确性都无法得到保证。

// bad

{list.map((item, index) => <Foo key={index} {...item} />)}

// good

{list.map(item => <Foo key={item.id} {...item} />)}

5.20建议在Refs里使用回调函数

// good

<Foo ref={ref => this.fooRef = ref} />

// not so good

<Foo ref=”fooRef” />