本规范本着五项原则去做约束：

1、可理解性：其他人可以接手代码并理解它的意图和一般途径，无需原开发人员的完整解释。

2、直观性：代码中的东西一看就能明白，尽管其操作过程复杂。

3、可适应性：代码以一种数据上的变化不要求完全重写方法。

4、可扩展性：在代码架构上可对核心功能的扩展。

5、可调式性：出错时，代码可以给你足够的信息来直接确定问题所在

痛点：

1. 目前项目中代码写法不统一，风格差异比较大。

规范: 针对目前的问题，对项目中的写法做出如下约定（这里不做规范而是约定）：

* 1.1 强制使用统一缩进
* 1.2 针对css/html命名和使用做出统一约定
* 1.3 针对JS命名做出统一约定
  1. **强制使用统一缩进**

1.1.1代码块的子成员距离块顶部强制两个空格缩进。

1.1.2 等号赋值操作：’=’ 号前后各增加一个空格 var a = 1

1.1.3 对象赋值操作：’:’ 号前面增加一个空格,花括号开始和结束各增加一个空格

如：var a = { b: 1 }

1.1.4 定义数组：中括号开始结束各增加一个空格，多个成员‘，’之后增加一个空格

如：var arr = [ “name” ]; var arr = [ “name”, “age” ];

1.1.5 连续成名多个变量，强制’,’之后使用空格或换行,换行格式：必须保持与前一个变量具有相同的缩进

如： var a = 1, b = 2;

如： var a = 1,

b = 2;

1.1.6 定义函数： 定义函数时强制在名 称前，’()’前后,增加一个空格

如： function fn () {}

使用函数：‘（）’前后不允许出现空格

如：fn();

1.1.7 条件语句的定义： 强制在 ‘()’前后增加空格,条件语句或变量前后增加空格, \

else if、else 前不增加空格

如：if ( a = 1 ) { console.log( 1 ) }else if ( a = 2 ) { console.log(2) }else { console.log(3)}

1.1.8 强制在分号之后增加空格：var a = 1; var b = 2;

1.1.9 强制在圆括号内使用统一空格：

如：foo( ‘123’ ); var x = ( 1+ 2 ) \* 3; if ( a ) {}

1.1.10 注释规范， HTML/CSS单行注释 内容前后增加一个空格，JS单行注释双斜杠前增加一个空格， HTML/CSS/JS 多行注释 必须换行并在内容之前增加两个空格

* 1. **针对css/html命名和书写做出统一**约定**（**约定**的目的是使，内容、结构、表现、行为进行解耦）**

1.2.1 class类名优先使用 block\_\_element\_modifier 命名法则

1.2.2 子元素class 命名要继承父级的命名空间

1.2.3 无论子元素嵌套层级多深 只能继承最上层的父元素命名空间

1.2.4 样式表写法约束1：不能使用 id或者tag选择器

1.2.5 样式表写法约束2：CSS选择器查找最多不能超过2层

1.2.6 html约束1： 统一使用两个空格代替制表符

1.2.7 尽量遵循HTML标准和语义，任何时候都尽量使用最少的标签并保持最小的负杂

度

1.2.8 严禁内联样式

1.2.9 多使用具有语义的标签，严禁通篇使用<div><span>标签。

如：一个列表应该使用 <ul>

1.2.10 严禁使用纯样式标签 如： <b>

**1.3 针对JS命名做出统一**约定

1.3.1：命名空间： 小驼峰

1.3.2：构造函数： 大驼峰

1.3.3：常量： 全大写并单下划线做连词符

1.3.4 私有变量： 单下划线+小驼峰

1.3.5 私有变量（常量）：单下划线+全大写并单下划线做连词符

1.3.6 命名必须符合语义化，严禁出现 名称与实际不符，或者没有语义的命名

例如：具有行为的：动词+名词，如：getResume,goHome

具有某种状态的： (名词或者动词)+Status， 如 ResumeStatus, LoginStatus

布尔值类型的： is+（名词或者动词）， 如 isLogin, isSubmit

对缩进和命名约定的说明：可读性与代码作为文本文件的格式化方式有关。可读性大部分内容都是和缩进相关的，当所有人都使用一样的缩进方式时，整个项目中的代码都会更加易于阅读。命名亦然，合适的命名和科学的命名方式都是提升可读性的方式。

痛点：

1. 老项目中代码可读性非常差，针对可读性做一下规范

* 2.1 html 注释规范
* 2.2 css 注释规范
* 2.3 js 注释规范

**html 注释规范**

2.1 头部增加多行注释：使用关键字格式 Create by name on Date

如：<!--

Create by Allen.sun on 2019/10/22

-->

如果：该页面为多人协作则需要加上Collaborator关键字，多个协作者之间用顿号连接

如：<!--

Create by Allen.sun on 2019/10/22

Collaborator：zhangsan、lisi、wanger、

-->

2.2 代码块注释：写明模块的功能模块：单人示例2.2.1、多人示例2.2.2

2.2.1如： <!-- 导航模块 --> 注意中杠 开始和结束处各增加一个空格

2.2.2如： <!-- 导航模块by Allen.sun -->

2.3 待完成的代码块注释：TODO: + 模块名称 ，单人示例2.3.1、多人示例2.3.2TODO：表示待完成，有多名协作者时可起到提示作用，这个模块处于待完成状态

2.3.1如： <!-- TODO:导航模块 -->

2.3.2如： <!-- TODO:导航模块by Allen.sun -->

注意事项： 模块完成后需要手动去TODO字段，告诉协作者这个模块已经完成

2.4 迭代注释使用多行注释：使用关键字格式 Modified by name on Date，如果多人开

如：我修改了导航模块

<!-- 导航模块 -->

<!--

Modified by Allen.sun on 2019/10/22

Note: 修改了tab切换的默认样式

-->

注意：不需要结束注释标签，因为该注释就在模块之内生效，结束标签会提高浏览器的载荷

2.5 页面中原则上不允许出现大段注释的代码，如有需求需要使用关键字格式 Destoried by name Date ，并且该注释要处于 模块注释之上，协作者一目了然

如：

<!--

Destoried by Allen.sun on 2019/10/22

Module: 导航模块

Note: 暂时不用下个版本使用

-->

<!-- 导航模块 -->

**css 注释规范**

2.1 声明规范：统一使用上面提到的两个字段 Create/Collaborator 并使用多行注释，格式同上,：星号是对齐的

/\*!

\* Create by Allen.sun on 2019/10/22

\* Collaborator: zhangsan、lisi、wangwu

\* Module：NavMod

\* Route /page/css/index/css

\*/

2.2 CSS分块注释：采用单行注释 多人协作 注明作者

如: /\* 导航栏样式 \*/

如: /\* 导航栏样式 by Allen.sun\*/

如未完成的样式: /\* TODO: 导航栏样式 \*/

2.3 迭代注释：采用多行注释

如：

/\* 导航栏样式 \*/

/\*!

\* Modified by Allen.sun on 2019/10/22

\* Note: 修改了主题配色

\*/

2.4 页面中原则上不允许出现大段注释的代码，如有需求需要使用关键字格式 Destoried by name Date ，并且该注释要处于 模块注释之上，协作者一目了然

如：

/\*！

\* Destoried by Allen.sun on 2019/10/22

\* Module: 导航模块

\* Note: 暂时不用下个版本使用

\*/

/\* 导航模块 \*/

**js 注释规范**

**声明：本规范只是覆盖到了 代码块的创建和文本的说明，更详细的规则可参见痛点3的详细规范；**

3.1 js 声明注释和html/css注释格式相同

3.2 js 声明迭代注释html/css注释格式相同

3.3 js 大段代码块被注释和html/css注释格式相同

3.4 js 关于代码块注释单独进行说明：

1、原则上如果是遵循本规范开发的代码命名具有语义化只需要进行单行注释

如：// 获取简历数据

如：// 获取简历数据 by Allen.sun

2、对返回值参数做单独注释：

如： // args0: XX，args1:xx 返回值：viod 0

3、如果进行老代码进行迭代或者修改某些功能，需要在修改过得代码头部添加多行注释，强制添加Note，内容自定义。目的是写明逻辑帮助后人，并用作己用。

如：

/\*\*

\* Modified by Allen.sun on 2019/10/22

\* Note: 这段代码是用来处理数据库格式的，注意类型的差异

\*/

关于注释规范的说明：本注释的规范不可能全部覆盖所有情况，注释的字段内容是可扩展的，但是要保证 html/css/js的易用性，也就是做到大体统一。记起来也比较好记。关于js的注释说明，js相对于 html/css 是更可变的，各种各样的情况的注释还需要细化。注释的重要性自不必多说，他是前人和后人沟通的桥梁，书写注释也需要注意，表达简单明了，抓住关键核心，不做赘述。

痛点：

1. js代码可读性差，维护性差。结构不清晰，耦合度高，文件分包随意导致的整个文件体系的混乱不堪。

* 3.1 详细的提升可读性的方案
* 3.2 提升可维护性：代码降维，方法行数做限定。
* 3.3 优化代码结构
* 3.4 代码耦合度降低合理封装不做过度封装，函数适应性加强
* 3.5 提高文件分包的科学性。

**3.1 详细的提升可读性的方案**

3.1.1 每一函数或方法都要包含一个规范的注释（声明注释和参数返回值注释），过程 注释单独作要求：

1、在方法中如果有大段代码用于处理某一项单独的任务需要任务描述注释，单行注释即可

如： // Description：本段代码适用于计算数据之间的差异

1. 方法中如果出现常量则需要注释说明常量的意义

如：// Sign：用于计算的padding值

1. 如果出现复杂的算法，或者使用非常规的方法去完成某一项任务， 则需要进行单号注释，说明思路。帮助他人进行理解也防止自己 遗忘。

如： // Description: 该算法用于计算元素处于页面的精确高度，采用递归的方式进行计算，如果没有父级则停止运算

4、变量创建时即指定他的类型：无论使用注释的方式还是匈牙利命 名法亦或是类型注释，都可以有效的提升代码的可读性也更容易 定位产生错误的原因

单行注释：

var name = “zhangsan”; // string类型

var num = 0; // number类型

匈牙利命名法：

var sName = “zhangsan”;

Var nNum = 0;

Var bIsLogin = false;

Var oResumeData = {};

这种使用缩写前缀的方式，不用写注释即可为后来者提示到该变量的类型

使用类型注释

Var resumeNumber /\* :Number \*/ = 1;

Var name /\* :String \*/ = “zhangsan”

**3.2提升可维护性：代码降维，方法行数做限定。**

3.2.1 代码降维

条件语句的判断在实际中的使用： if 看起来没有问题 if...else 看起来问题也不大。那么if...else if...else..if...else 呢？我只是在文档中写这一串就感觉头晕了！更可况需要在这样的流水账逻辑中看没有任何注释的大段代码了。

这种情况还是维度比较低的耦合，实际工作中经常会看到if...else...嵌套if...else又嵌套个if...else 这样的代码是令人头疼且比较LOW的，所以代码逻辑降维是非常有必要去强行规范的。关于这点做如下规范：

1. 代码中禁止使用if...else...if 逻辑
2. If...else 嵌套禁止超过两层
3. If逻辑嵌套禁止超过三层
4. 条件判断代码总层数不得超过三层

3.2.2 方法逻辑

在实际项目中，方法不得超过100行，（方法中已经被拆分成。实际想限制到更小的函数或者包含大量的变量的函数做特殊情况），原因是真正在开发项目中，几乎没有特别复杂的逻辑需要大段的算法实现，一般都是面向过程的逻辑开发，适度的封装非常有利于减少代码行数、提升可读性、可维护性、复用性。

**3.3 优化代码结构**

3.3.1 代码结构有条理，思路要清晰

1、杜绝使用全局变量，

2、变量声明在同一的局部作用域的最上部

如：(function () {

var a = 1,

b = 2,

c = 3,

d = 4,

const name = null;

function get () {

Let a = 1,

B = 2,

C = 3

}

})()

上面示例了错误的命名方式

3、统一类型，变量、对象、方法，归类统一写在一起方便阅读和注释

4、简化引用：

多层对象引用时要确定指针和值

var data = res.data.Data,name = data.name;

特别在是包含循环的代码块中可以改善迭代效率，并提升可读性。

1. 定时器的使用
   1. 如果代码中需要使用异步去完成某一项功能，那么定时器要放在代码块的最后去使用。正常业务逻辑中如果突然插入一个定时器，即使你知道他是一个异步的不影响下面的逻辑，在代码中有这个一大段也是很影响可读性的

**3.4 代码耦合度降低合理封装不做过度封装，函数适应性加强**

1、降低代码的耦合度 -- 松散耦合

（1）注意对象的耦合 ：var a = {}, b = a, a.num = 1; 使用时注意引用引起的后果是可预知的。

（2）html/js 的耦合，HTML中禁止直接在DOM上添加内联事件，禁止HTML中添加script标签中直接写JS代码

（3）html/css的耦合，诸如 el.style.color = “red”这种写法是禁止的，因为CSS是空值页面样式显示的而不是JS。可以使用松散耦合 el.className = ‘color\_red’

2、上文中无数次提到过封装，封装的重要的不言而喻。但是也要防止过度封装减少过度封装给可读性带来的影响，同时也会影响你工作中的开发效率。 不要为了封装而封装。

**3.5 提高文件分包的科学性**

1、目前项目中包含多个脚本包，多个视图包，多个静态文件包。导致了文件重复，脚本，样式，视图文件乱入，查找起来十分困难。项目中的引用也乱七八糟，想删除重复的文件都存在未知的风险。

针对这种情况前端需要约定一个合理的目录结构，以后开发严格按照新规则去分包。

1. 文件的命名应该见明知意。文件的名字应当包含项目的名字并用“-”来进行修饰

临时记录：CSS样式杜绝内联样式，修改样式采用prolly补丁的方式进行覆盖。Js清除冗余代码