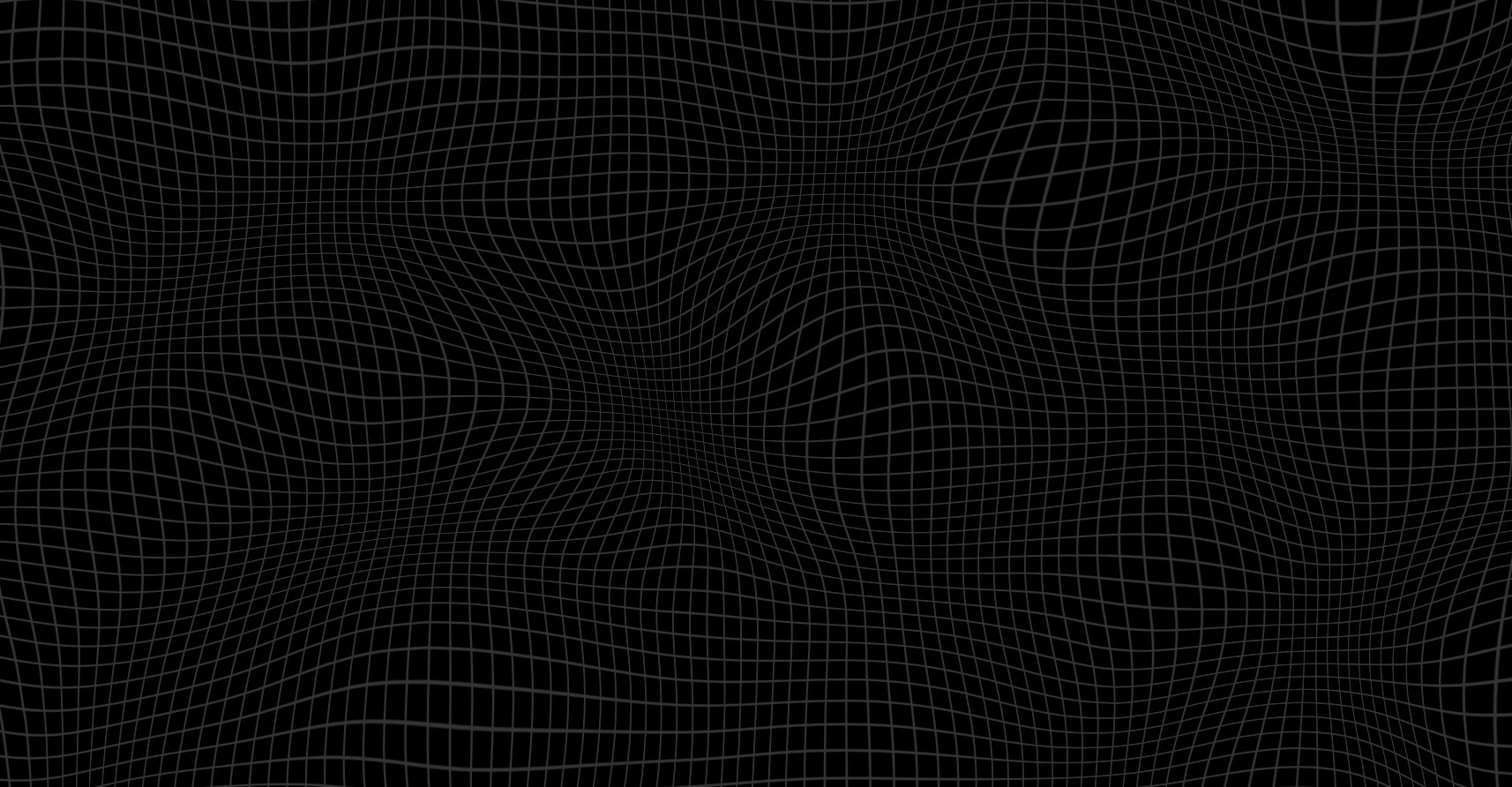


****

**工作室代码开发文档**

**适用于JAVA开发人员丶前端**

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 版本 | 作者 | 更新日期 | 备注 |
| 1 | 1.0.0 | 汪志 | 2018/06/07 | 初稿 |

**特别说明：最新更新的规范一般为黄色底纹标示。**

# 编程规范

1. **命名规范**
2. 【强制】所有编程相关命名均不能以下划线或美元符号开始，也不能以下划线或美 元符号结束。

错误示范：\_name / \_\_name / $Object / name\_ / name$ / Object$

1. 【强制】所有编程相关的命名严禁使用拼音与英文混合的方式，更不允许直接使用 中文的 方式。说明：正确的英文拼写和语法可以让阅读者易于理解，避免歧义。注意， 即使纯拼音命名方式也要避免采用。

错误示范：Dazhe = 打折 / Caiwu = 财务 / Userguanli = 用户管理

正确示范：youku / taobao / userManagement 等国际识别拼音可认为是英语

1. 【强制】类名使用 UpperCamelCase 风格，必须遵从驼峰形式，但以下情 形例外：

（领域模型的相关命名）PO / DTO / VO / DAO 等。

正确示范：MarcoPolo / UserPO / XmlService / TcpUdpDeal / TaPromotion

错误示范：macroPolo / UserPo / XMLService / TCPUDPDeal / TAPromotion

1. 【强制】方法名、参数名、成员变量、局部变量统一使用UpperCamelCase 风格，  
    必须遵守驼峰形式，首字母小写。

正确示范：localValue / getHttpMessage() / inputUserId

1. 【强制】常量命名全部使用大写，单词用\_隔开，单词必须是有意义易阅读的。

正确示范：REDIS\_USER\_LOGIN\_TOKEN\_KEY

1. 【强制】抽象类使用abstract开头；异常处理类使用Exception结尾；测试类使用 Test结尾；Service类使用“I”开头，实现类使用“Impl”结尾；

正确示范：AbstractUser / ServiceException / UserManagerTest / IUserService / UserServiceImpl

1. 【强制】POJO 类中的任何布尔类型的变量，都不要加 is，否则部分框架解析会 引起序列化错误。

错误示范：定义为基本数据类型 boolean isSuccess；的属性，它的方法也是 isSuccess()，RPC框架在反向解析的时候，“以为”对应的属性名称是 success，导致属性获取不到，进而抛出异常。

1. 【强制】包名统一使用小写，点与点之间只有一个自然单词。

正确示范：com.xxx.xxx.utils = com/公司名/项目名/模块

1. 【强制】接口类中的方法和属性不要加任何修饰符号（public 也不要加），保持 代 码的简洁性，并加上有效的注释。尽量不要在接口里定义变量，如果一定要定义变 量，肯定是与接口方法相关，并且是整个应用的基础常量。

正确示范：UserPO getUserList();

1. 【强制】枚举名称请使用Enum结尾，枚举成员名称需要全大写，单词间用下划 线隔开。说明：枚举其实就是特殊的常量类，且构造方法被默认强制是私有。

正确示范：枚举名字：DealStatusEnum；成员名称：SUCCESS / UNKOWN\_REASON。

1. **【强制】各层命名规范如下：**
2. Service/Dao及项目层规范
3. 获取单个对象请使用get开头。
4. 获取集合对象请使用get开头，list结尾。
5. 获取统计类数据的方法请使用count开头。
6. 插入的方法请使用save开头。
7. 删除方法请使用remove开头。
8. 修改方法请使用update开头。
9. Service 接口使用“I”+“业务名”+“Service”组合，

正确示范：IUserManagementService。

1. Service实现类请使用“业务名”+“Service”+“Impl”组合，如UserManagementServiceImpl。
2. Dao层的类一致使用Dao结尾，接口实现规则与Service保持一致。
3. Dao的映射配置文件请使用“Dao名称”+“Mapper”结尾，

正确示范：Dao类是IUserDao，映射文件为 UserDaoMapper 。

1. service层项目里的operate/property/smartcard/commons各种项目数据业务逻辑实现应按照文件归属来命名如：

正确示范：service-equip服务的com.xxx.xxx.property.service

#IPropertyEquipService，接口 I+“文件所属”+“业务”+“Service” 组合，实现类“文件所属”+“业务”+“Service”+“Impl”。

1. 领域模型规范
2. 数据对象：xxxPO，xxx对应数据表名。
3. 数据传输对象：方法参数大于5个时请使用DTO进行传输，DTO按照

当前方法的业务操作命名，规则如：项目名+模块名+业务名+DTO，如果是分页对象那么请在DTO前加上ByPage。

正确示范：AppUserGetUserRoomsInfoDTO

1. 展示对象：xxxVO，xxx对应的为展示的网页名称。
2. POJO为以上领域模型的统称，禁止使用POJO取名。

领域模型释义：主要用于远程调用等需要大量传输对象的地方。  
比如我们一张表有100个字段，那么对应的PO就有100个属性。  
但是我们界面上只要显示10个字段，客户端用WEB service来获取数据，没有必要把整个PO对象传递到客户端，这时我们就可以用只有这10个属性的DTO来传递结果到客户端，这样也不会暴露服务端表结构.到达客户端以后，如果用这个对象来对应界面显示，那此时它的身份就转为VO。

1. **【强制】系统常量命名规范：**
2. Redis
3. 缓存的key应按照 “REDIS”+”服务名”+”业务操作”+”CACHE”,

正确示范: REDIS\_USER\_GET\_USER\_INFO\_CACHE

1. 缓存的时间应按照“REDIS”+“服务名”+“业务操作”+“CACHE”+

“TIME”。

正确示范：REDIS\_USER\_GET\_USER\_INFO\_CACHE\_TIME

1. RabbitMQ
2. Queue应按照“RABBITMQ”+“服务名”+“业务操作”+“QUEUE”

正确示范: RABBITMQ\_USER\_SAVE\_USER\_INFO\_QUEUE

1. RoutingKey按照 “RABBITMQ”+“服务名”+“业务操作”+“ROUTING\_KEY”

正确示范: RABBITMQ\_USER\_SAVE\_USER\_INFO\_ROUTING\_KEY

1. Exchange 按照“RABBITMQ”+“服务名”+“Exchange”

正确示范: RABBITMQ\_USER\_EXCHANGE

1. Platform 此类的常量通常为通用型常量定义，如短信账号配置，系统秘钥/公钥配置等，尽量保证字面意思清晰完整。
2. 【强制】工具类命名规范，无论是时间丶分页丶图片等工具类都要以Util结束。

正确示范：JsonUtil

1. 【强制】方法体内的参数如果大于等于4个时，参数换行排列。

正确示范：

**public** Page<OperateGroupPO> getGroupPage(  
 String currentPage,  
 String groupNum,  
 String groupName,  
 String groupStatus) {  
 **return null;**  
 }

1. 【强制】父类（除接口，虚类）请用Base开头，如果是接口用IBASE开头。
2. 【强制】Controller类必须依照页面创建，比如：系统设置模块下有角色管理，

组织管理，权限管理等，那么Controller就需要建立对应的三个类。

RequestMapping路由编写如下：

正确示范：

RequestMapping(“setting/group/”)

RequestMapping(“setting/role/”)

RequestMapping(“setting/function/”)

1. 【强制】如果包以property丶operate丶smartcard等形式分包的，那么该下面

所有的类必须要加上前缀。

正确示范：PropertyXXXXXDao

1. **OOP规范**
2. 【推荐】Object的equals()方法很容易抛出NullException，应该确定有值或常量 来调用equals()方法。

正确示范：”test”.equals(object)

错误示范：object.equals(“test”)

推荐使用：java.util.Objects$equals() 自JDK1.7引入

1. 【强制】所有相同对象的类型请使用equals()方法来比较。

说明：Integer a = ? 在-128 ~127之间赋值，Integer是在IntegerCache.cache产生的，可以复用对象，因此可以直接使用==进行比较，但是如果赋值超过此区间之后的数据全部会再堆中产生，这是一个巨型大坑，推荐使用equals()方法来比较。

1. 【强制】所有类中的构造函数方法中严禁增加任何代码实现逻辑，如有初始化数据 请写在init()方法中。
2. 【推荐】使用String的split方法时，请检查最后一个分隔符后是否还有内容，否则 有抛出IndexOutOfBoundsException的风险。
3. 【推荐】循环体内如遇字符串进行拼接时，请使用StringBulider/或StringBuffer ， 前者是线程非安全后者是线程安全的，根据实际情况进行最优的选择。

错误示范：

String word = **"a"**;

**for** (**int** i = 0 ; i < 10; i++){  
 word = word + **"b"**;

}

说明：String对象在循环时都会new一个StringBulider对象，然后进行append()方法来追加，这样会造成不必要的内存消耗。

1. 【推荐】Controller层接收参数如果大于5个时，必须使用DTO进行接收传输，

返回数据时如果页面所需的只是单表数据，用PO返回和传输，如果页面所需的数

据来自多张表请转化VO或者MAP显示。

1. **控制语句**
2. 【强制】 switch语句块内，每个case语句都要有break/return返回，或者注释说明 接下来会 进入到哪个case语句中；在每个switch语句块中都必须要包含一个default， 即使没有代码。
3. 【推荐】尽量少用else，if-else的方式可以改写成如下：

**if**(b){  
 ....  
 **return** obj;  
 }

1. 【强制】即使if语句中只有一句代码也不能省略掉{}花括号。

错误示范：**if** (**"a"**.equals(word)) **return "a"**;

1. 【推荐】循环语句中请勿超过超过3层，如果业务逻辑复杂且集合过多，请考虑

其他方式实现。

1. 【推荐】循环体中要考虑代码的性能，以下操作请移至循环外处理；如定义对象

丶变量丶获取连接丶进行不必要的try-catch。

1. **注释规范**
2. 【强制】成员变量：单个成员或代码块处，请直接使用 “//”，PO实体的成员

丶代码关键逻辑点必须注释。

1. 【强制】方法：方法的注释包括描述丶作者丶创建时间丶参数列表丶返回说明,

为了代码的简洁只在Service/Dao层的接口加上该注释即可，因Controller层没有接口所以加在方法中。

正确示范：

*/\*\*  
\* <desc>  
\* 查询用户列表信息。  
\* <desc/>  
\** ***@param userId*** *用户Id  
\** ***@return*** *用户列表*

*\** ***@author*** *Wenchao.L  
\** ***@createDate*** *2017/08/14*

*\*/* UserPO getUserList(Long userId);

1. 【强制】类：类的注释包括描述丶创建时间。

正确示范：

*/\*\*  
 \* <desc>  
 \* 用户管理接口  
 \* </desc>  
 \** ***@createDate*** *2017/08/14  
 \*/* **public interface IUserManagementService**{}

1. 【强制】常量类中所有的常量必须用“//”加上描述。

# 日志异常

1. **异常处理**
2. 【强制】不要捕捉Java类库中集成自RuntimeException的运行时异常，

如NullException/IndexOutOfBoundsException，这类异常应该在代码内去规避掉。

来保证代码的健壮性。

1. 【强制】如try块里有涉及到事务操作，那么在cache语句块中就注意手动

回滚事务。

正确示范：

**try**{  
 ...  
 }**catch** (Exception ex){  
 TransactionAspectSupport.*currentTransactionStatus*().setRollbackOnly();  
 }

1. 【强制】finall语句块中一定切记要释放资源丶如需要也可加上try-catch。
2. 【强制】不能再finally语句块中return结果，否则finally返回结果后，

try语句块的return则不会执行，容易造成混淆。

1. 【推荐】如代码不够稳定必须加上try-catch语句块处理。
2. **日志规约**
3. 【强制】应用中不能直接使用日志系统（log4j丶Logback）中的API，而应

依赖使用日志框架中的API，使用门面模式的日志框架，有利于各个类的日志处

理方式统一。

正确示范：

**private static final** Logger ***log*** = Logger.*getLogger*(Abc.**class**);

1. 【强制】应用中如果catch语句块中捕捉到异常记录的正确格式如下

正确示范：

log.error(**"【项目名\_模块名\_方法】:获取用户列表失败"**,Throwable);

1. 【推荐】应用中如果在重要的方法中，建议关键操作节点不仅要try-catch

还要有每步的日志记录下，以便于追踪丶调试丶观察等。

1. 【推荐】输出的POJO类必须重写toString方法，否则输出的是类的hashcode

不具任何参考价值。

1. 【推荐】谨慎使用INFO记录日志不要滥用，否则会造成磁盘的消耗，ERROR只

是用于记录代码逻辑错误，异常等严重的错误，如非必要，请不要在此场景下使用

ERROR打出此日志。

# MYSQL规约

1. **建表规范**
2. 【强制】表达是与否概念的字段，使用is\_xxx的方式命名，数据类型可以为unsigned 丶int(1)丶等（1表示是，0表示否）。

说明：任何字段如果有负数，一律用unsigned类型。

1. 【强制】表名丶字段名必须使用小写字母开头；禁止出现数字丶大写字母开头，

由于修改表名的代价比较大，所以建表时请谨慎取有意义的表名。

正确示范：s\_user\_info / s\_order / s\_task13\_config

错误示范：13\_task\_config / S\_user\_info / s\_1\_order

1. 【强制】表名以及字段名都不允许驼峰方式命名，单个单词请使用“\_”隔开。
2. 【强制】小数类型请使用decimal方便以后扩展，禁止使用double丶float。
3. 【强制】varchar是可变长字符，不预先分配空间，长度不要超过5000，如果存储

长度大于此长度，请使用TEXT字段来进行存储并将此字段移出此表，新建一张存储

TEXT字段的表用主键关联，以免影响其他字段索引效率。

1. 【强制】每张表一定是必备四个字段 id丶create\_time丶update\_time丶status。

说明：id=主键，create\_date=数据创建时间，update\_time=数据修改时间，status=

数据是否逻辑删除，status默认值 1，表达式为 0删除 1 可用 2禁用

1. 【强制】每张表字段的comment必须要加上有意义的注释。
2. 【推荐】表的命名最好使用 “业务名称+表的作用”，以免容易与其他模块混淆。

正确示范：tiger\_task / user\_role

1. 【强制】每张表的字段都需要是NOT NULL，而不是NULL，因为有的索引会不 支持有NULL的列，有些业务场景产生的数据如果不能满足所有列都有值的，请初 始空字符或其他。
2. **SQL规范**
3. 【推荐】在count某字段时，请不要使用count(列名)代替count(\*)，因为count(\*)

会统计字段为null的，而count(列名)则不统计null的。

1. 【强制】禁止使用存储过程，存储过程难以调试，和扩展。
2. 【强制】不得使用外键主键建立关联关系，一切关系使用逻辑标识建立，以免不

的资源消耗。

1. 【推荐】in操作能规避掉就尽量规避掉，如果业务需要实在难以规避，则需评估

in里的集合数据的量，尽量控制在1000个以内。

1. 【强制】使用ISNULL()来判断是否为NULL值；NULL与任何值比较都为NULL。

说明：

1. NULL<>NULL 的返回结果是NULL，不是false。
2. NULL=NULL返回的结果是NULL，不是true。
3. NULL<>1返回的结果是NULL，而不是true。

# 前端规范

1. **vue 开发命名规范**
2. 【强制】views 下的文件夹命名：
3. views 下面的文件夹代表着模块的名字；
4. 尽量由名词组成；正确示范：car、order、cart 错误示范：greet、 good
5. 单词只能有一个；正确示范：car、order、cart 错误示范：carInfo 、carpage
6. 以小写开头；正确示范：car 错误示范：Car
7. 【强制】views 下的vue文件命名：

1）views 下面的 vue 文件代表着页面的名字；

2）放在模块文件夹之下；

3）只有一个文件的情况下不会出现文件夹，而是直接放在 views 目录下面，如 Login Home；

4）尽量是名词；

5）大写开头，开头的单词就是所属模块名字；正确示范：CarDetail、CarEdit、CarList

6）名字至少两个单词；正确示范： CarDetail 错误示范： Car

7）常用结尾单词有（Detail、Edit、List、Info、Report）

8）以 Item 结尾的代表着组件；正确示范： CarListItem、CarInfoItem

3. 【强制】vue 方法命名：

A. vue方法放置顺序：

1）components

2）props

3）data

4）created

5）mounted

6）activited

7）update

8）beforeRouteUpdate

9）metods

10）filter

11）computed

12）watch

B. method 自定义方法命名:

1）动宾短语

正确示范：jumpPage、openCarInfoDialog

错误示范：go、nextPage、show、open、login

2）ajax 方法以 get、post 开头，以 data 结尾

正确示范：getListData、postFormData

错误示范：takeData、confirmData、getList、postForm

3）事件方法以 on 开头

正确示范：onTypeChange、onUsernameInput

4）init、refresh 单词除外

5）尽量使用常用单词开

正确示范：set、get、open、close、jump

6）驼峰命名

正确示范：getListData

错误示范：get\_list\_data、getlistData

C.data props 方法注意点

1）使用 data 里的变量时请先在 data 里面初始化

2）props 指定类型，也就是 type

3）props 改变父组件数据 基础类型用 $emit ，复杂类型直接改

4）ajax 请求数据用上 isLoading、isError 变量

5）不命名多余数据，现在是详情页、你的数据是 ajax 请求的，那就直接声明一个对象叫 d，而不是每个字段都声明

6）表单数据请包裹一层 form

D. 生命周期方法注意点

1）不在 mounted、created 之类的方法写逻辑，取 ajax 数据

2）在 created 里面监听 Bus 事件

1. **一般规范**

1. 【强制】文件命名总是以字母开头而不是数字。

2. 【强制】资源的字母名称必须全为小写，说明：因为在某些对大小写字母敏感的操作系统中，当文件通过工具压缩混淆后，或者人为修改过后，大小写不同而导致引用文件不同的错误，很难被发现。

3. 【强制】需要对文件增加前后缀或特定的扩展名（比如 .min.js, .min.css），抑或一串前缀（比如 3fa89b.main.min.css）。这种情况下，建议使用点分隔符来区分这些在文件名中带有清晰意义的元数据。

正确示范：my-script.js / my-camel-case-name.css / i-love-underscores.html / thousand-and-one-scripts.js / my-file.min.css

错误示范：MyScript.js / myCamelCaseName.css / i\_love\_underscores.html / 1001-scripts.js / my-file-min.css

1. 【强制】注释内容建议代码为什么要这么写，背后的考量是什么。也可以加入所思考问题或是解决方案的链接地址。
2. **HTML规范**
3. 【强制】推荐使用 HTML5 的文档类型申明： <!DOCTYPE html>；

2. 【强制】建议使用 text/html 格式的 HTML。避免使用 XHTML。XHTML 以及它

的属性，比如 application/xhtml+xml 在浏览器中的应用支持与优化空间都十分有限；

3. 【强制】HTML 中最好不要将无内容元素的标签闭合；

正确示范：<br>

错误示范：<br />

4. 【强制】建议使用能通过标准规范验证的 HTML 代码，除非在性能优化和控制文件大小上不得不做出让步。

正确示范：

<!DOCTYPE html>

<meta charset="utf-8">

<title>Test</title>

<article>This is only a test.</article>

错误示范：

<title>Test</title>

<article>This is only a test.

5. 【强制】推荐脚本异步加载，说明：一段脚本放置在 <head> 内，比如 <script src="main.js"></script>，其加载会一直阻塞 DOM 解析，直至它完全地加载和执行完毕。这会造成页面显示的延迟。特别是一些重量级的脚本，对用户体验来说那真是一个巨大的影响。

正确示范：

<html>

<head>

<link rel="stylesheet" href="main.css">

</head>

<body>

<!-- body goes here -->

<script src="main.js" async></script>

</body>

</html>

6. 【强制】根据元素被创造出来时的初始意义来使用它。比如用 heading 元素来定义头部标题，p 元素来定义文字段落，用 a 元素来定义链接锚点，等等。说明：有根据有目的地使用 HTML 元素，对于可访问性、代码重用、代码效率来说意义重大。

正确示范：

<header>

<h1>My page title</h1>

</header>

<main class="news-page" role="main">

<section class="page-section news">

<header>

<h2 class="title">All news articles</h2>

</header>

<article class="news-article">

</article>

</section>

</main>

错误示范：

<b>My page title</b>

<div class="news-page">

<div class="page-section news">

<div class="title">All news articles</div>

<div class="news-article">

</div>

</div>

</div>

7. 【强制】对页面上的媒体而言，像图片、视频、canvas 动画等，要确保其有可替代的接入接口。图片文件我们可采用有意义的备选文本（alt），视频和音频文件我们可以为其加上说明文字或字幕。

1. **JavaScript规范**

1. 【强制】需要创建一个新的封闭的定义域，那就用 IIFE（立即执行的函数表达式）。可以避免干扰，也使得内存在执行完后立即释放。所有脚本文件建议都从 IIFE 开始。

正确示范： (function(){}());

错误示范：(function(){})();

2. 【强制】ECMAScript 5 严格模式可在整个脚本或独个方法内被激活。推荐在独立的 IIFE 中应用它。说明：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。

正确示范：

(function(){

'use strict';

// Your code starts here

}());

错误示范：

'use strict';

(function(){

// Your code starts here

}());

3. 【强制】总是使用 var 来声明变量。说明：如不指定 var，变量将被隐式地声明为全局变量，这将对变量难以控制。

正确示范： var x = 10, y = 100;

错误示范：x = 10;y = 100;

1. 【强制】在简单的循环语句中加入函数是非常容易形成闭包而带来隐患的。

5.【强制】只在对象构造器、方法和在设定的闭包中使用 this 关键字。说明：this 的语义容易误导。它时而指向全局对象（大多数时），时而指向调用者的定义域（在 eval 中），时而指向 DOM 树中的某一节点（当用事件处理绑定到 HTML 属性上时），时而指向一个新创建的对象（在构造器中），还时而指向其它的一些对象（如果函数被 call() 和 apply() 执行和调用时）。

6.【强制】用三元操作符分配或返回语句。

正确示范：return x === 10 ? 'valid' : 'invalid';

错误示范：

if(x === 10) {

return 'valid';

} else {

return 'invalid';

}

1. **CSS和Sass(Scss)规范**

1. 【强制】ID和class(类)名总是使用可以反应元素目的和用途的名称，或其他通用名称。代替表象和晦涩难懂的名称。

正确示范：

.heavy {

font-weight: 800;

}

.important {

color: red;

}

错误示范：

.fw-800 {

font-weight: 800;

}

.red {

color: red;

}

2. 【强制】省略“0”值后面的单位。

正确示范：

padding-bottom: 0;

margin: 0;

错误示范：

padding-bottom: 0px;

margin: 0em;

3. 【强制】使用连字符（中划线）分隔ID和Class（类）名中的单词。

正确示范：

#video-id {}

.ads-sample {}

错误示范：

.demoimage {}

.error\_status {}

4. 【强制】这是一个选择器内书写CSS属性顺序的大致轮廓。这是为了保证更好的可读性和可扫描重要。

作为最佳实践，我们应该遵循以下顺序（应该按照下表的顺序）：

结构性属性：

display

position, left, top, right etc.

overflow, float, clear etc.

margin, padding

表现性属性：

background, border etc.

font, text

5. 【强制】为了保证一致性和可扩展性，每个声明应该用分号结束，每个声明换行。

正确示范：

.test {

display: block;

height: 100px;

}

错误示范：

.test {

display: block; height: 100px

}

6. 【强制】每个选择器和属性声明总是使用新的一行。

正确示范：

h1,

h2,

h3 {

font-weight: normal;

line-height: 1.2;

}

错误示范：

a:focus, a:active {

position: relative; top: 1px;

}

1. 【强制】选择器嵌套 (SCSS) ,使代码变得更清洁和可读。
2. 【强制】嵌套选择器和CSS属性之间空一行。
3. 【强制】上下文媒体查询(SCSS),在Sass中，当嵌套选择器时也可以使用上下文媒体查询。生成的CSS将被转换，媒体查询将包裹选择器的形式呈现。有助于保持媒体查询属于的上下文。

10.【强制】嵌套顺序和父级选择器(SCSS)

当使用Sass的嵌套功能的时候，重要的是有一个明确的嵌套顺序，以下内容是一个SCSS块应具有的顺序。

当前选择器的样式属性

父级选择器的伪类选择器 (:first-letter, :hover, :active etc)

伪类元素 (:before and :after)

父级选择器的声明样式 (.selected, .active, .enlarged etc.)

用Sass的上下文媒体查询

子选择器作为最后的部分

# **SVN基本原则**

## 代码

1. 【强制】项目中的.idea丶.iml丶target包的个人文件禁止提交，以免影响他人本地 开发环境。
2. 【强制】config-service 项目的dev环境的后缀为properties配置文件一般为本地配 置文件，由于每个人配置的端口丶账号丶密码都会不一致，所以一旦修改不得提交影响 他人开发，如有公共配置不得不提交的请备份好本地修改后还原并修改提交。
3. 【强制】所有项目里yml涉及到公共配置的也不得随意修改并提交，如有必要请知 会领导。
4. 【强制】提交代码前请review一下提交的文件，确保不会出现错误提交。
5. 【建议】修改文件时尤其是这个文件可能会被他人修改时，请遵循update丶lock丶 set丶unlock丶commit的步骤，尽量避免merge带来的不必要麻烦。
6. 【强制】严禁随意修改公共方法和他人代码，如有必要请知会相关人。
7. 【建议】在不影响他人环境编译丶运行报错前提下，建议每天提交一次代码，最多 不超过两天。