

**G03组代码规范**



G03小组：

成员：李俊 林鑫 胡锦波

**目录**

[1. 标识符命名规范 3](#_Toc502068826)

[1.1. 概述 3](#_Toc502068827)

[1.1.1. 统一 3](#_Toc502068828)

[1.1.2. 达意 3](#_Toc502068829)

[1.1.3. 简洁 4](#_Toc502068830)

[1.1.4. 英文 vs 拼音 4](#_Toc502068831)

[1.2. 页名 4](#_Toc502068832)

[1.3. 局部变量名 4](#_Toc502068833)

[2. 代码格式 5](#_Toc502068834)

[2.1. 代码块格式 5](#_Toc502068835)

[2.1.1. 缩进风格 6](#_Toc502068836)

[2.1.2. 空格的使用 6](#_Toc502068837)

[2.1.3. 空行的使用 7](#_Toc502068838)

[3. 注释规范 8](#_Toc502068839)

[3.1. 注释vs代码 8](#_Toc502068840)

[3.2. 块级别注释 8](#_Toc502068841)

[3.3. 3.4 行内注释 8](#_Toc502068842)

[4. 最佳实践和禁忌 8](#_Toc502068843)

[4.1. 每次保存的时候，都让你的代码是最美的 8](#_Toc502068844)

[4.2. 每个if while for等语句，都不要省略大括号{} 8](#_Toc502068845)

[4.3. 在需要留空的地方放一个空语句或注释，告述读者，你是故意的 9](#_Toc502068846)

[4.4. 减少代码嵌套层次 9](#_Toc502068847)

[4.5. 程序职责单一 11](#_Toc502068848)

[4.6. 多做些单元测试类 11](#_Toc502068849)

# 标识符命名规范

## 概述

标识符的命名力求做到统一、达意和简洁。尽量做到每个人按照规范来，多人开发如一人开发一样。

### 统一

统一，是指对于同一个概念，在程序中用同一种表示方法，比如对于注册，既可以用register，也可以用login，但是我们只能选定一个使用，至少在一个页面中保持统一。

如果对同一概念有不同的表示方法，会使代码混乱难以理解。即使不能取得好的名称，但是只要统一，阅读起来也不会太困难，因为阅读者只要理解一次。

### 达意

达意，是指标识符能准确的表达出它所代表的意义，准确有两成含义，正确和丰富。

例如：使用数组时，按序命名为array1、array2等。

### 简洁

简洁，是指在统一和达意的前提下，用尽量少的标识符。如果不能达意，宁愿不要简洁。不要使用省略元音的缩写方式。

例如：accountPassword 太长， accoutPswd则较好，但是apd就不好了。

### 英文 vs 拼音

尽量使用通俗易懂的英文单词，避免拼音与英文混用。

例如：表示注册，用register比较好,用enroll则不好，用zhuce尚可接受。

## 页名

使用小写字母，单词间不要用字符隔开。

例如：使用register、successful等

同时对于每一页来说，应该包含四类文件，分别是控制页面能干什么的.js、控制标题格式的.json、描述页面包含哪些内容的.wxml、描述页面格式的.wxss，如下图



## 局部变量名

参数和局部变量名首字母小写，使用骆驼法则。尽量不要和域冲突，尽量表达这个变量在方法中的意义。

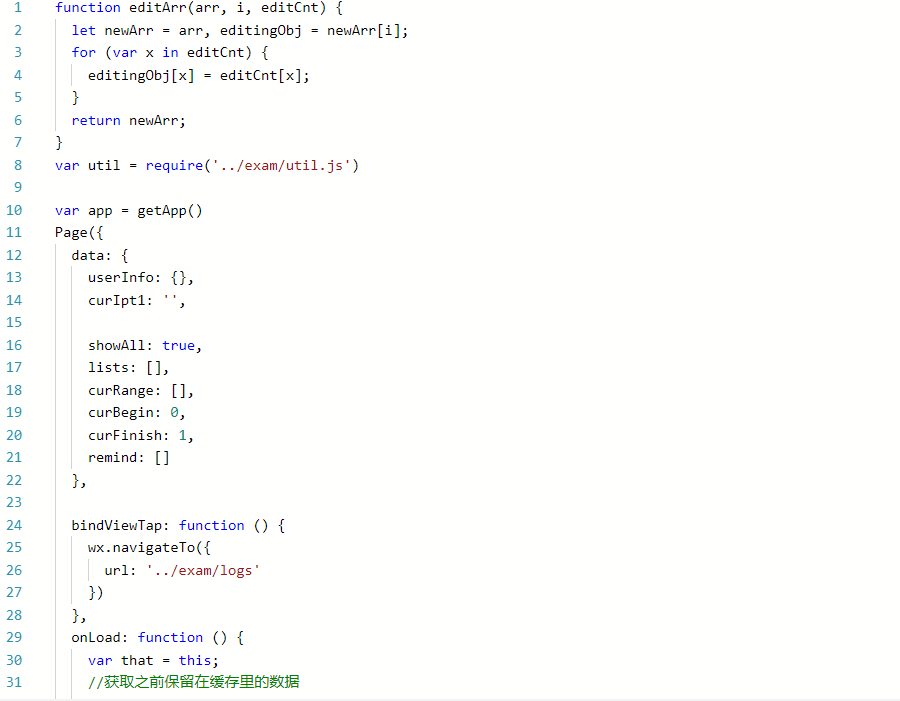
# 代码格式

使用tab缩进源代码。使用alt+shift+f来格式化代码。

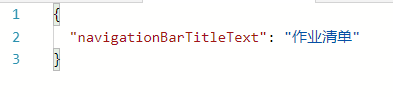
注：格式化代码后还需手动来调下。

## 代码块格式

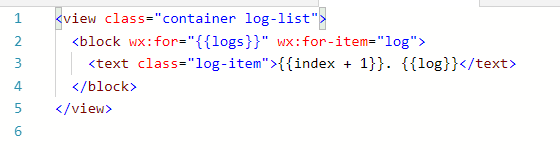
.js格式



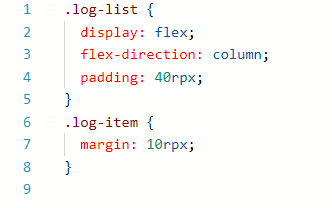
.json格式



.wxml格式



.wxss格式



### 缩进风格

大括号的开始在代码块开始的行尾，闭合在和代码块同一缩进的行首，同一层次的代码要保持整齐。

### 空格的使用

#### 表示分割时用一个空格

不能这样：

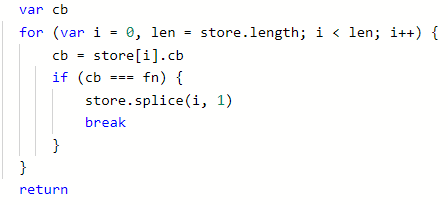
**if** ( a > b ) {

*//do something here*

};

#### 运算符两边用一个空格隔开

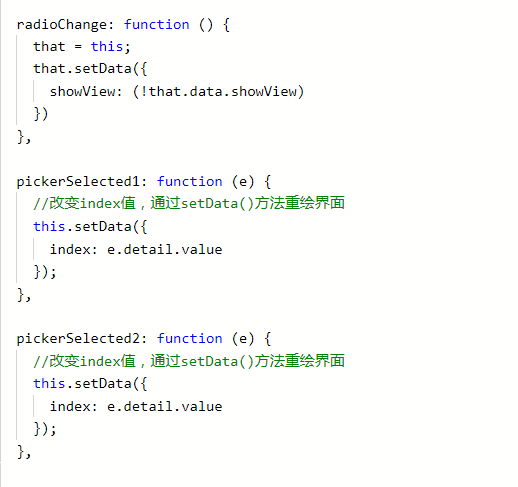
例如：



#### 逗号语句后如不换行，紧跟一个空格

### 空行的使用

空行可以表达代码在语义上的分割，注释的作用范围，等等。将类似操作，或一组操作放在一起不用空行隔开，而用空行隔开不同组的代码，如图：



上例中的空行，使注释的作用域很明显.

* 连续两行的空行代表更大的语义分割。
* 方法之间用空行分割（尽量用一行空行）
* 域之间用空行分割
* 超过十行的代码如果还不用空行分割，就会增加阅读困难

# 注释规范

## 注释vs代码

（1）注释宜少而精，不宜多而滥，更不能误导。

（2）命名达意，结构清晰， 类和方法等责任明确，往往不需要，或者只需要很少注释，就可以让人读懂；相反，代码混乱，再多的注释都不能弥补。所以，应当先在代码本身下功夫。

（3）不能正确表达代码意义的注释，只会损害代码的可读性。

（4）过于详细的注释，对显而易见的代码添加的注释，罗嗦的注释，还不如不写。

（5）注释要和代码同步，过多的注释会成为开发的负担。

## 块级别注释

单行时用 //, 多行时用 /\* .. \*/。较短的代码块用空行表示注释作用域，较长的代码块要用

/\*------ start: ------\*/

和

/\*-------- end: -------\*/

包围。

## 3.4 行内注释

行内注释用 // 写在行尾。

# 最佳实践和禁忌

## 每次保存的时候，都让你的代码是最美的

程序员都是懒惰的，不要想着等我完成了功能，再来优化代码的格式和结构，等真的把功能完成，很少有人会再愿意回头调整代码。

## 每个if while for等语句，都不要省略大括号{}

看下面的代码：

**if** (a > b)

a++;

如果在以后维护的时候，需要在a > b 时，把b++，一步小心就会写成：

**if** (a > b)

a++;

b++;

这样就错了，因为无论a和b是什么关系，b++都会执行。 如果一开始就这样写：

**if** (a > b) {

a++;

}

相信没有哪个笨蛋会把b++添加错的。而且，这个大括号使作用范围更明显，尤其是后面那行很长要折行时。

## 在需要留空的地方放一个空语句或注释，告述读者，你是故意的

比如：

**if** (!exists(order)) {

;

}

或：

**if** (!exists(order)) {

*//nothing to do*

}

## 减少代码嵌套层次

代码嵌套层次达3层以上时，一般人理解起来都会困难。下面的代码是一个简单的例子：

**public** **void** demo(**int** a, **int** b, **int** c) {

**if** (a > b) {

**if** (b > c) {

doJobA();

} **else** **if** (b < c) {

doJobB()

}

} **else** {

**if** (b > c) {

**if** (a < c) {

doJobC();

}

}

}

}

减少嵌套的方法有很多：

* 合并条件
* 利用 return 以省略后面的else
* 利用子方法

比如上例，合并条件后成为：

**public** **void** demo(**int** a, **int** b, **int** c) {

**if** (a > b && b > c) {

doJobA();

}

**if** (a > b && c > b) {

doJobB();

}

**if** (a <= b && c < b && a < c) {

doJobC();

}

}

如果利用return 则成为：

**public** **void** demo(**int** a, **int** b, **int** c) {

**if** (a > b) {

**if** (b > c) {

doJobA();

**return**;

}

doJobB()

**return**;

}

**if** (b > c) {

**if** (a < c) {

doJobC();

}

}

}

利用子方法，就是将嵌套的程序提取出来放到另外的方法里。

## 程序职责单一

关注点分离是软件开发的真理。人类自所以能够完成复杂的工作，就是因为人类能够将工作分解到较小级别的任务上，在做每个任务时关注更少的东西。让程序单元的职责单一，可以使你在编写这段程序时关注更少的东西，从而降低难度，减少出错。

## 多做些单元测试类

单元测试类可以提前发现代码中的好多问题。