# 项目编码规范

## 一、项目描述

该项目是“开饭啦”线上点餐系统，对编码规范中，我们前端页面用WebStorm编写。后端用软件PyCharm编写。前端用到的技术是vue,后端用Python编写。

二、Python编码规范

**1、字符缩紧**，每级四个空格(一个Tab)：空格和制表符避免混用（python3 不允许）可以用pycharm reformat code去格式化，变成一个格式

**2、if 语句**，多个条件（条件块），无明确对if内的编码规范，只对条件块

（1）多个条件缩进换行

简化写法可以不需要：

no\_str\_date\_cols = [i[0] for i in df.dtypes if i[1] != 'string' and i[1] != 'timestamp']

# 不增加额外的缩进

if (this\_is\_one\_thing and

that\_is\_another\_thing):

do\_something()

（2）if前面添加注释，为支持高亮编辑器提供区分（内部代码也需要）

# 添加一行注释，这将为支持语法高亮的编辑器提供一些区分

if (this\_is\_one\_thing and

that\_is\_another\_thing):

# 当两个条件都是真，我们将要执行

do\_something()

**3、在换行条件语句前**，增加额外缩进，推荐（pycharm换行后自动增加额外缩进，产生 \ ，其他编辑器是否支持）

if (this\_is\_one\_thing

and that\_is\_another\_thing):

do\_something()

**4、多行结构**，list进行换行，结束花括号/圆括号/中括号应该是最后一行的第一个非空字符

my\_list = [

1, 2, 3,

4, 5, 6,

]

或者是最后一行第一个字符

my\_list = [

1, 2, 3,

4, 5, 6,

]

**5、最大行长度**：限制所有航79个字符（可以用编辑器自动格式化进行自适应编辑器屏幕显示），亦可以适当增大至100

换行时采用反斜杠 \ (多个 with 语句）

**6、换行应该在二元操作符(+ - \* /等）**，无明确规定，但是为保证可读性，前后保证一致

建议在操作符后换行

# 好的: 易于匹配操作数和操作符

income = (gross\_wages

+ taxable\_interest

+ (dividends - qualified\_dividends)

- ira\_deduction

- student\_loan\_interest)

**7、空格**

用两行空行分割顶层函数（下面可能有内部函数）和类的定义.

类内方法的定义用单个空行分割（方法签名换行）.

额外的空行可被用于(保守的(sparingly))分割一组相关函数(groups of related functions).（多个函数结果相互引用）

在一组相关的单句中间可以省略空行（方法体，相关代码行：如main方法）

在函数中使用空行时,请谨慎的用于表示一个逻辑段落(indicate logical sections).

**8、源文件编码**

Python核心发布中的代码应该始终使用UTF-8（或Python2中用ASCII）。

文件使用ASCII（Python2中）或UTF-8（Python3中）不应有编码声明。

**9、导入**

# 通常应该在单独的行中导入(Imports)

import os

import sys

# 这样也是可以的

from subprocess import Popen, PIPE

排列顺序：标准库（io,os,json),第三方，本地应用（项目间引用）

分为决定导入，相对导入

import mypkg.sibling

from mypkg import sibling

from mypkg.sibling import example

**10、字符串引号**

支持单引号，双引号，但是不建议混用

当字符串包含单双引号是，使用另一种引号来避免字符串种使用 \ 转义 (可以单双引号联合使用）

**11、表达式和语句中的空格**

# 紧挨着圆括号,方括号和花括号的（括号周围无空格）

spam(ham[1],{eggs: 2})

# 紧贴在逗号,分号或冒号前的(逗号，分号，冒号前面无空格）

if x == 4: print x, y; x, y = y, x

# 在切片中冒号像一个二元操作符，冒号两侧应该有相同数量的空 格（把它看作最低优先级的操作符）。

# 在一个扩展切片中，两个冒号必须有相等数量的空格。

# 例外：当一个切片参数被省略时，该空格也要被省略。

ham[1:9], ham[1:9:3], ham[:9:3], ham[1::3], ham[1:9:]

ham[lower:upper], ham[lower:upper:], ham[lower::step]

#扩展切片，两个冒号必须有相等数量空格

ham[lower+offset : upper+offset]

ham[: upper\_fn(x) : step\_fn(x)], ham[:: step\_fn(x)]

ham[lower + offset : upper + offset]

# 紧贴着函数调用的参数列表前开式括号(open parenthesis )的

spam(1)

# 紧贴在索引或切片(slicing?下标?)开始的开式括号前的

dct['key'] = lst[index]

# 在赋值(或其它)运算符周围，不要为了与其他的赋值（或其它） 操作符对齐，使用不止一个空格。

x = 1

y = 2

long\_variable = 3

**12、其它建议**

避免尾随空格。（不要在一行代码写完添加多余空格，一些编辑器不支持）

因为它通常是无形的,它可能会让人很困惑:如一个反斜杠,后跟一个空格和一个换行符不算作一行延续标记。有些编辑器不保护它，并且许多项目(如CPython本身)导向钩子,拒绝它。

始终在这些二元运算符两边放置一个空格:赋值(=), 比较(==, <, >, !=, <>, <=,>=, in, not in, is, is not), 布尔运算 (and, or, not).

**13、注释**

同代码不一致的注释比没注释更差.当代码修改时,始终优先更新注释!（方法修改时，优先修改，使用def搜索）

注释应该是完整的句子. （英文）如果注释是一个短语或句子,首字母应该大写, 除非他是一个以小写字母开头的标识符(永远不要修改标识符的大小写).

如果注释很短,最好省略末尾的句号。注释块通常由一个或多个由完整句子构成的段落组成,每个句子都应该以句号结尾.

你应该在句末的句号后使用两个空格,以便使Emacs的断行和填充工作协调一致 (译按:应该说是使这两种功能正常工作,". "给出了文档结构的提示).

用英语书写时,断词和空格是可用的.

非英语国家的Python程序员:请用英语书写你的注释,除非你120%的确信 这些代码不会被不懂你的语言的人阅读.

注释块

注释块通常应用于跟随着一些(或者全部)代码并和这些代码有着相同的缩进层次.

注释块中每行以'#'和一个空格开始(除非他是注释内的缩进文本)，以相同的

注释块内的段落以仅含单个'#'的行分割.

行内注释

行内注释应该谨慎使用.

一个行内注释是和语句在同一行的注释.行内注释应该至少用两个空格和语句分开. 它们应该以'#'和单个空格开始.

**14、文档字符串**

应该一直遵守在PEP 257中编写好的文档字符串(又名"docstrings")的约定。

**15、版本注记**

如果你要将RCS或CVS的杂项(crud)包含在你的源文件中,按如下做.

\_\_version\_\_ = "$Revision$"

# $Source: E:/cvsroot/python\_doc/pep8.txt,v $

这个行应该包含在模块的文档字符串之后,所有代码之前,上下用一个空行分割.

**16、命名约定**：保持内部一致性(小写字母，\_)

Python库的命名约定有点混乱,所以我们将永远不能使之变得完全一致--- 不过还是有公认的命名规范的.

新的模块和包(包括第三方的框架)必须符合这些标准,但对已有的库存在不同风格的, 保持内部的一致性是首选的.

根本原则

用户可见的API的公开部分的名称，应该体现用法而不是实现。

**17、描述:命名风格**（小写字母单词，'\_')

有许多不同的命名风格.以下的有助于辨认正在使用的命名风格,独立于它们的作用.

以下的命名风格是众所周知的:

b (单个小写字母)

B (单个大写字母)

小写串 如:getname

带下划的小写串 如:\_getname,lower\_case\_with\_underscores

大写串 如:GETNAME

带下划的大写串 如:\_GETNAME,UPPER\_CASE\_WITH\_UNDERSCORES

CapitalizedWords(首字母大写单词串) (或 CapWords, CamelCase/驼峰命名法 -- 这样命名是由于它的字母错落有致的样子而来的.这有时也被当作StudlyCaps. 如:GetName

**18、命名约定**

a.避免的名字

永远不要用字符"|",'O',"I'作变量名(某些字体种，字符不能与数字1,0分开，当要使用'I'时，用'L'代替它。

b.包名，类名，模块名，全局变量名，函数名，异常名

包：简短，小写，不鼓励下划线

模块名：全小写名字，可以在模块种使用下划线来提高可读性（可以保留现在，首字母大写，同类名一样：若改，改动较多，同其他编码习惯）

类名：首字母大写

c.全局变量名：(要在类成员方法，内部方法定义全局变量（类似goto,跳来跳去, 即使可以），可读性差

与用于函数的约定差不多，被设计可用'from M import \*'来使用那些模块（具体函数），应该使用all机制导出全局变量，或使用加前缀的旧规则，在那些不想被导入

全局变量（还有内部函数和类）前加一个下划线

d.函数名：

总是小写，单词之间用下划线（'\_')分割增加可读性

e.异常名：因为异常应该是类，同类的规范，为见名知义，后缀加上ERROR

f.函数和方法参数：

总是使用self(或cls)做类方法（实例方法，成员方法：即非静态方法）的第一个参数

如果一个函数的参数名与保留关键字冲突，最好是为参数名添加一个后置下划线而不是使用缩写或错误的拼写（如class\_ 比 clss好。（使用同义词来避免更好）

g.方法名和实例变量：避免特殊命名（尽量不要带下划线），特殊引用（可以在本类再次声明）

大体核函数相同：使用小写单词，必要时使用下划线（'\_') 分隔，增加可读性

仅为不打算作为类的公共接口的内部方法和实例变量使用一个前导下划线

为了避免和子类命名冲突，使用两个前导下划线调用Python的名称改编规则

如 Python用类名改编这些名字：如果类Foo有一个属性名为 \_\_a,通过Foo.\_\_a不能访问，通常，两个前导下划线用来避免与子类的属性名冲突。

执著用户：可以通过Foo.\_Foo\_\_a来访问

h.常量：

通常在模块级别定义（类名上面），并且所有字母大写，单词用下划线分开

如：DATA\_DIR, RUNING\_MODE

i.继承的设计

确定类的方法和实例变量（统称为：“属性”）是否公开，如果有疑问，选择非公开；之后把其变成公开比把一个公开属性改成非公开要容易

公开属性：期望与其他用户共享；非公开不希望被共线；你不保证非公开属性不会改变甚至被删除

Python 没pirvate,因为无真正私有的属性（java private）

属性的另一个类别是"API子集"的一部分（java protected：可以在当前包及其子类引用）

Pythonic 的指南：

公开属性没有前导下划线。如果和保留关键字冲突，给属性名添加一个后置下划线，这比缩写和拼错更可取，同类方法命名建议

注1：参见上面对类方法的参数名的建议。

对于简单的公开数据属性，最好只暴露属性名，没有复杂的访问器/修改器方法。记住，Python为未来增强提供了一条简单的途径，你应该发现简单的数据属性需要增加功能行为。在这种情况下，使用属性来隐藏简单数据属性访问语法后面的功能实现。

注1：特性仅工作于新风格的类（现在保持原样，保留一致）。

注2：尽量保持功能行为无副作用，尽管副作用（打印日志，做不相干的处理）如缓存通常是好的。

注3：计算开销较大的操作避免使用特性，属性注解使调用者相信访问（相对）是廉价的。

如果确定你的子类会被子类化，并有不想子类使用的属性，考虑用两个前导下划线'\_\_' 无后置下划线'\_'来命名他们。这将调用Python的名称改编算法，类名讲被改编成属性名。当子类不无意间包括相同的属性名时，这有助于帮助避免属性名冲突

注1.注意改编名称仅用于简单类名，如果一个子类使用相同类名和属性名，仍然会有命名冲突

注2.名称改编会带来了一定不便，如调试核getattr(),但是名称改编算法有良好的文档，也容易手工执行

注3：不是每个人都喜欢名称改编。尝试平衡避免意外的名称冲突和高级调用者的可能。

18.公共和内部接口：

任何向后兼容保证 只适用公共接口。此举能让用户清楚区分公开内部接口

文档接口被认为是公开，除非文档明确声明是临时、内部接口，免除通常的向后兼容保证

所有非文档化的接口应假定为内部接口。

**19、程序设计建议**

a.与单值, 比如None比较, 使用is或is not，不要用等号操作符。

b.使用 is not 操作符而不是 not...is。虽然这两个表达式的功能相同，前者更具有可读性并且更优

c.当实现有丰富的比较的排序操作时，最好实现所有六个操作符（\_\_eq\_\_，\_\_ne\_\_，\_\_lt\_\_，\_\_le\_\_，\_\_gt\_\_，\_\_ge\_\_）而不是依靠其它代码只能进行一个特定的比较

为了减少所涉及的工作量，functools.total\_ordering()装饰器提供了一个工具来生成缺失的比较函数。

d.总是使用def语句而不是使用赋值语句绑定lambda表达式到标识符上,对于逻辑处理多的也是如此，如map里面有过得逻辑处理,不要多次换行。

风格良好：

def f(x): return 2\*x

风格不良：

f = lambda x: 2\*x

e.从Exception而不是BaseException中派生出异常。直接继承BaseException是保留那些捕捉几乎总是错的异常的。

设计异常层次结构基于区别，代码可能需要捕获异常，而不是捕获产生异常的位置。

f.适当使用异常链。Python3中，“raise X from Y”用来表明明确的更换而不失去原来追踪到的信息。

当故意替换一个内部异常（Python 2中使用“raise X”而Python 3.3+中使用“raise X from None”），

确保相关的细节被转移到新的异常中（比如当转换KeyError为AttributeError时保留属性名，或在新的异常消息中嵌入原始异常的文本）

g.Python 2中产生异常，使用raise ValueError('message')替换老的形式raise ValueError,'message'。

后一种形式是不合法的Python 3语法,python 3仍然是 raise ValueError('message')

h.捕获异常时，尽可能提及特定的异常，而不是使用空的except：子句(如模块导入是ImportError)。

try:

import platform\_specific\_module

except ImportError:

platform\_specific\_module = None

空的except：子句将捕获SystemExit和KeyboardInterrupt异常，这使得很难用Control-C来中断程序，也会掩饰其它的问题。

限制使用空‘except’子句两种情况的最佳实践：

1.打印异常日志：如果异常处理程序将打印或记录跟踪；至少用户将会意识到有错误发生。

2.try...finally善后:如果代码需要做一些清理工作，但是随后让异常用raise抛出。处理这种情况用try...finally更好。

i：当用一个名字绑定捕获异常时，更喜欢Python2.6中添加的明确的名称绑定语法，这是Python3中唯一支持的语法，并避免与旧的基于逗号的语法有关的歧义问题：

try:

process\_data()

except Exception as exc:

raise DataProcessingFailedError(str(exc))

捕获操作系统异常时，优先使用 Python 3.3引进的明确的异常层次，通过errno值(错误编码)自省。

j.另外：避免掩盖错误，尽量在try..except 写较少代码（过多无法知道谁引发异常）：

风格良好：

try:

value = collection[key]

except KeyError:

return key\_not\_found(key)

else:

return handle\_value(value)

风格不良：根据Python 异常层次进行明确异常日志

try:

# 很多代码

return handle\_value(collection[key]) #有可能不确定是获取以key为下标的值抛异常，还是通过handle\_value报错

except KeyError:

# 捕获由handle\_value()抛出的KeyError

return key\_not\_found(key)

k:当一个资源是一个局部特定的代码段，它使用后用with语句确保迅速可靠的将它清理掉。也可以使用try/finally语句。

无论何时做了除了获取或释放资源之外的一些操作，都应该通过单独的函数或方法调用上下文管理器：

风格良好，开启事务（上下文管理器）：

with conn.begin\_transaction():

do\_stuff\_in\_transaction(conn)

风格不良：

with conn（未在with中调用上下文管理器）:

do\_stuff\_in\_transaction(conn)

后者例子未提供任何信息表明 enter和exit方法做了什么，除事务结束后关闭连接。此情况下，明确很重要

返回语句保持一致。函数中所有返回语句都有返回值，或都没有返回值。

如任意一个返回语句有返回值，那么其他任意没有返回值的返回语句（可能是多路条件分支都有返回值：switch case )应该明确return None

并且一个显示的返回语句应该放在函数结尾（有可能所有条件分支都没进入）

风格良好：

def foo(x):

if x >= 0:

return math.sqrt(x)

else:

return None

def bar(x):

if x < 0:

return None

if x ==1:

return None

return math.sqrt(x)

风格不良：

def foo(x):

if x >= 0:

return math.sqrt(x)

def bar(x):

if x < 0:

return

return math.sqrt(x)

l:使用字符串方法代替String模块：效率更高

字符串方法总是更快并且与Unicode 字符串使用相同API，如要兼容Python2.x，忽视该原则

使用".startswith()"和".endswith()"代替字符串切片来检查前缀和后缀。

startswith()和 endswith()更清晰，并且减少错误率：

如：

风格良好：if foo.startswith('bar'):

风格不良：if foo[:3] == 'bar': （人为干预过多，让API来做更高效，准确）

对象类型比较使用isinstance()代替至二级比较类型更高型：

风格良好：if isinstance(obj, int):

风格不良：if type(obj) is type(1):（人为干预过多，让API来做更高效，准确）

当检查一个对象是否是字符串时，牢记它也可能是一个Unicode 字符串！！！！，python2.x中，str和Unicode有共同基类：basestring.

所以你可以这么做：

if isinstance(obj,basestring):

但是：python3.x中，Unicode 和 basestring不在存在（只有str),并且字节对象不在是String的一种（而是一个integers序列）

对于序列（成员有序排列,可通过下标访问.），（字符串，列表，元组），理由空序列是false的事实：

风格良好：if not seq:

if seq:

风格不良: if len(seq)

if not len(seq)

不要书写依赖后置空格的字符串。这些后置空格在视觉上无法区分，并且有些编辑器（或最近，reindent.py)将去掉他们。

m:不要用 == 来将布尔值与True 或False进行比较

风格良好：if greeting:

风格不良：if greeting == True:

糟糕的：if greeting is True:

**20、函数注释：**

为了向前兼容,Python 3中的函数注释代码应该更好地使用PEP 484语法。(在前一节中有一些注释格式的建议：参考之前函数，代码注释规范)。

Python标准库应该对使用这些注释采取保守的态度,但允许在新的代码或大型重构中使用这些注释规则。

对于代码，如果想要用函数注释的不同用法，推荐使用如下格式的注释:

# type: ignore

在靠近文件的顶部,这告诉类型检查器忽略所有注释。(更详细的关闭类型检查器报错的方法，可以在 PEP 484 中查找)。

二、vue编码规范

**1、UI框架及css预处理器选择**

（1）PC端Vue项目UI框架：ElementUI（优先）、iView

（2）移动端Vue项目UI框架：mint-ui（优先）、vant

（3）sass/scss、less、stylus  
 推荐less：less相对比sass更简洁，而stylus的语法灵活性开放性过大， 不利于团队协作开发。

**2、脚手架、插件选择**

（1）脚手架vue-cli2.0 OR vue-cli3.0  
 推荐vue-cli2.0：团队中熟悉2.0版本脚手架的人居多，脚手架是一 个工具类的存在。  
 （2）网络请求：axios  
 （3）时间格式化插件：[momentjs](http://momentjs.com" \t "_blank)  
 （4）富文本编辑器：ueditor  
 （5）图片剪裁插件：[cropperjs](https://github.com/fengyuanchen/cropperjs" \t "_blank)  
 （6）图表：[echarts](http://www.echartsjs.com/index.html" \t "_blank)

**3、命名规范**

（1）id和class的命名原则

应反映该元素的功能或使用通用名称，而不要用抽象的晦涩的命名 （原则：见名知其义）

（2）文件夹、文件名、id、class具体命名规范:[驼峰命名法](https://baike.baidu.com/item/%E9%AA%86%E9%A9%BC%E5%91%BD%E5%90%8D%E6%B3%95/7794053?fr=aladdin" \t "_blank)

文件夹采用“大驼峰命名法”

文件名采用“大驼峰命名法”

id：采用“小驼峰命名法”  
 class：采用“中划线法命名法”

（3）id和class命名越精简越好，只要足够表达意思，这样有助于理解，同 时也能提高代码效率

（4）命名中尽量避免使用中文拼音，应该采用更简明有语义的英文单词进 行组合。

命名注意缩写，但是不能盲目缩写；

不允许通过1、2、3等序号进行命名；

避免class与id重名；

id注意用于标识模块或页面的某一个父容器区域，名称必须唯一， 不要随意新建id；

class用于标识某一个类型的对象，命名必须言简意赅；

尽可能提高代码模块的复用，样式尽量用组合的方式；

规则名称中不应该包含颜色、定位等与具体显示效果相关的信息， 应该用意义命名，而不是结果名称。

（5）Vue 项目中的命名

·Store 中的Module 使用“小驼峰命名法”

·Store 中的Mutation 使用 全部大写的下划线命名法

·Store 中的state/getters/action 使用“小驼峰命名法”

·组件必须使用“大驼峰命名法”命名法,

·引用组件时禁止使用别名，模板内组件标签名遵循html 标签命名 规范，或者使用组件名

·组件名必须避免使用Vue保留标签名（包括HTML标签和Vue 内部标签）

·组件文件和组件使用相同的名字

·前端路由路径使用全小写命名法

**4、排版规范**

（1）、代码格式

大括号的使用约定。

\* 如果是大括号内为空，则简洁地写成{}即可，不需要换行

例：{}

\* 非空代码块则:

（1）左大括号前不换行有空格；

（2）左大括号后换行；

（3）右大括号前换行；

（4）右大括号后还有else等代码则不换行;

（5）表示终止的右大括号后必须换行。

例：

methods: {

testFunc () {

console.log("测试方法");

var flag = false;

if (this.num === 0) {

flag = true;

} else {

flag = false;

}

}

}

}

2、左右小括号与中间字符之间不出现空格。

3、if/for/while/switch 等保留字与括号之间都必须加空格

任何二目、三目运算符的左右两边都需要加一个空格。

5、注释的双斜线与注释内容之间有且只有一个空格

6、方法参数在定义和传入时，多个参数逗号后边必须加空格。

7、在 if/else/for/while/do 语句中必须使用大括号。即使只有一行代码，避免采用 单行的编码方式:if (condition) statements;

5、项目目录结构

（1）项目初始目录

node\_modules // 项目依赖包文件夹

build // 编译配置文件，一般不用管

build.js

check-versions.js

dev-client.js

dev-server.js

utils.js

vue-loader.conf.js

webpack.base.conf.js

webpack.dev.conf.js

webpack.prod.conf.js

config // 项目基本设置文件夹

dev.env.js // 开发配置文件

index.js // 配置主文件

prod.env.js // 编译配置文件

index.html // 项目入口文件

package-lock.json // npm5 新增文件，优化性能

package.json // 项目依赖包配置文件

src // 我们的项目的源码编写文件

App.vue // APP入口文件

assets // 初始项目资源目录，回头删掉

logo.png

components // 组件目录

Hello.vue // 测试组件，回头删除

main.js // 主配置文件

router // 路由配置文件夹

index.js // 路由配置文件

tatic // 资源放置目录

（2） src目录配置

App.vue // APP入口文件

components // 组件文件夹

common // 共用组件文件夹

page // 我们的页面组件文件夹

home // 模块划分文件夹

user // 模块划分文件夹

Main.vue // 项目主入口文件

plugins // 项目配置文件夹

ajax // 网络请求配置文件

http.js // 网络请求封装文件

api // 常用工具文件夹

index.js // 自定义工具注册入口

ValideCard.js // 身份证验证规则文件

valideEmail.js // 邮箱验证规则

valideTel.js // 手机号证规则

url // 项目配置文件

index.js // 接口注册入口

url.js // 接口API自定义文件

main.js // 项目配置文件

router // 路由配置文件夹

ndex.js // 路由配置文件

style // scss 样式存放目录（待定，选用less或scss）

base // 基础样式存放目录

\_base.scss // 基础样式文件

\_color.scss // 项目颜色配置变量文件

\_mixin.scss // scss 混入文件

\_reset.scss // 浏览器初始化文件

scss // 页面样式文件夹

\_content.scss // 内容页面样式文件

\_index.scss // 列表样式文件

style.scss // 主样式文件

（3）CSS属性编写顺序（一般遵循显示属性 -> 自身属性 -> 文本属性 -> 其他属性的书写格式）

显示属性：display/list-style/position/float/clear...

自身属性（盒模型）：width/height/margin/padding/border

背景：background

行高：line-height

文本属性： color/font/text-decoration/text-align/text-indent/vertical-align/white-space/content...

其他：cursor/z-index/zoom/overflow...

CSS3属性：transform/transition/animation/box-shadow/border-radius

如果使用CSS3的属性，如果有必要加入浏览器前缀，则按照-webkit-/-moz-/-ms-/-o-/std的顺序进行添加，标准属性写在最后

**注：补充说明**

1、删除多余的空行

所有方法与方法之间空1行

所有代码块之间空1行

2、删除多余的注释

删除注释掉的代码

删除没有意义的注释

3、删除多余的方法

如果方法没有使用到，请删除它

如果方法没有执行任何业务逻辑，请删除它或者给出一定注释

4、删除未被使用的资源文件

5、添加必要的注释

所有自定义的方法需要给出注释

比较大的代码块需要给出注释

所有代码中出现的阿拉伯数字需要给出注释

程序中出现加密／解密 逻辑的操作地方，需要给出注释说明过程（无论是系统还是自定义）

整体代码风格需要统一

代码后面的“{” 不需要单独占用一行

逻辑运算符 与 代码之前空一格