前言：蓝鲸开发规范主要包括应用开发的代码，用户设计体验，安全和性能等的规范和建议，以下将用【必须】来表示开发工程中必须遵守的规范，其他均为建议。

PEP 8

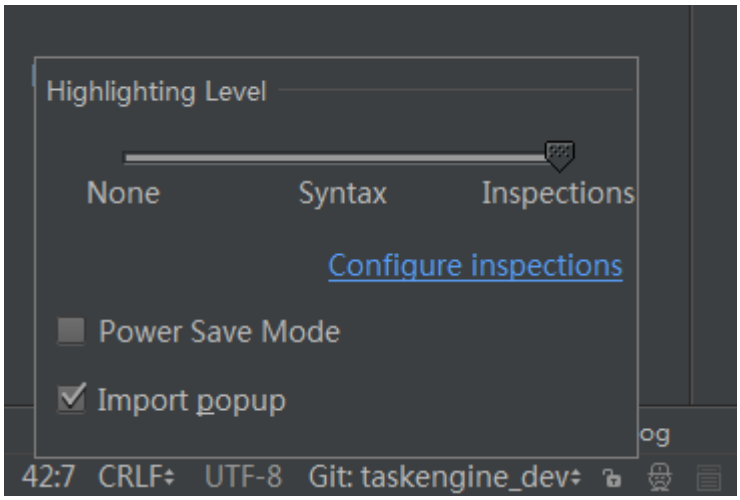
Python 编码规范：https://www.python.org/dev/peps/pep-0008/

- 1.Eclipse 中配置 PEP 8 代码提示

  将 PyDev 升级到高于 2.3.0 版本，打开 Window > Preferences > PyDev > Editor > Code

- 2.PyCharm 配置 PEP 8 代码提示

  直接在右下角调整 Highlighting Level 为 Inspections 就能自动 PEP 8提示

- 

- 3.建议修改在使用的 IDE 中修改 PEP8 的每行字数不超79字符规范，可修改为 Django 建议的 119 字符

  说明：其他编辑器或IDE配置请自行搜索

编码规范

【必须】代码编码

- 1.国际惯例，文件编码和 Python 编码格式全部为 utf-8 ，例如：在 Python 代码的开头，要统一加上 # -*- coding: utf-8 -*-。
- 2.Python 代码中，非 ascii 字符的字符串，请需添加u前缀

- ```
  # -*- coding: utf-8 -*-

  a = u"中国"
  ```

- 3.若出现 Python编 码问题，可按照以下操作尝试解决：

  在 Python 的安装路径中下的 /Lib/site-packages 下面创建文件 sitecustomize.py，内容如下

- ```
  import sys
  sys.setdefaultencoding('utf-8') # set default encoding as 'utf-8'
  ```

  如果没有加入该文件，则在有编码问题的 .py 代码中，加入以下代码：

- ```
  import sys
  reload(sys)
  sys.setdefaultencoding('utf-8')
  ```

Python编码规范

【必须】命名规范

- 1.包名、模块名、局部变量名、函数名

  全小写+下划线式驼峰

  示例：this_is_var

- 2.全局变量

  全大写+下划线式驼峰

  示例：GLOBAL_VAR

- 3.类名

  首字母大写式驼峰

  示例：ClassName()

- 4.变量名命名

  尽量体现变量的数据类型和具体意义
  ```

注：

- 变量名、类名取名必须有意义，严禁用单字母
- 变量名不要用系统关键字，如 dir type str等等

建议：

- bool变量一般加上前缀 is_ 如：is_success

【必须】 import 顺序

- 标准库
- 第三方库
- 项目本身
- (之间空行分隔)

注：

- 尽量不要引用

【必须】 models 内部定义顺序

- 1.All database fields
- 2.Custom manager attributes
- 3.class Meta
- 4.def (**str**)
- 5.def save()
- 6.def get_absolute_url()
- 7.Any custom methods

异常捕获处理原则

- 1.尽量只包含容易出错的位置，不要把整个函数 try catch
- 2.对于不会出现问题的代码，就不要再用 try catch了
- 3.只捕获有意义，能显示处理的异常
- 4.能通过代码逻辑处理的部分，就不要用 try catch

- 5.异常忽略，一般情况下异常需要被捕获并处理，但有些情况下异常可被忽略，只需要用 log 记录即可，可参考一下代码：

```
from contextlib import contextmanager

@contextmanager
def ignored(*exceptions):
    try:
        yield
    except exceptions as e:
        logger.warning(e)
        pass
index = 0
with ignored(ValueError):
    index = int(dic['operate'])
```

return early原则

提前判断并 return，减少代码层级，增强代码可读性

```
if not condition:
    return

# a lot if code
```

Fat model， thin view

逻辑代码和业务代码解耦分离，功能性函数以及对数据库的操作定义写在 models 里面，业务逻辑写在 view 里面。


@login_required
@check_section_role_by_ccid()
@xy_execute_task_lock()
def execute_task(request, biz_cc_id):...
@login_required
@check_section_role_by_ccid()
def re_execute_task(request, biz_cc_id):...
@login_required
@check_section_role_by_ccid()
def reset_task_data(request, biz_cc_id):...
@login_required
@check_section_role_by_ccid()
def re_create_task(request, biz_cc_id):...
@login_required
@check_section_role_by_ccid()
def reset_task_parameters(request, biz_cc_id):...
Fat view
class Task_single(models.Model):
...
def __unicode__(self):
...
class Meta:
...
Thin model

改为


def get_task_result(request, task_id, biz_cc_id):
task_ins = TaskFlowInstance.objects.get(business__cc_id=biz_cc_id,
pk=task_id, is_deleted=False)
task_result = task_ins.get_task_result(request)
result = {'result': True}
result.update(task_result)
return render_json(result)
Thin view
class TaskFlowInstance(models.Model):
@property
def template(self):...
def get_next_flow(self):...
def next_flow(self, request):...
def get_task_info(self):...
def get_task_template_tree(self):...
def fill_params(self, request, task_name, para
def get_bpm_task_constant(self, request):...
def get_bpm_task_params(self, request):...
def create_and_run_bpm_task(self, request):...
def get_task_result(self, request):...
def finish_task(self, request):...
def report_tag_data(self, request):...
def clone(self, request):...
def user_has_perm(self, user, flow_list):...
Fat model

权限校验装饰器异常抛出问题

建议权限不足时直接抛出异常，可以使用 django 自带的：

from django.core.exceptions import PermissionDenied

权限不足时抛出异常 PermissionDenied，之后应该返回什么样的页面由 handler 或者中间件去处理

分 method 获取 request 参数问题

一般可以分method 获取request参数，这样能够使代码更可读，且之后修改 method 时不必每个参数都修改

```
args = request.GET if request.method == "GET" else request.POST
business_name = args.get('business_name', '')
template_name = args.get('template_name', '')
```

使用数字、常量表示状态

两种的话改为 true/false，多种改为 enum 可读性更好

```
def enum(**enums):
    return type("Enum", (), enums)

StatusEnum = enum(
    SUCCESS=True,
FAIL=False,
)
```

其他注意问题

- 1.【必须】去除代码中的 print，否则导致正式和测试环境 uwsgi 输出大量信息
- 2.逻辑块空行分隔
- 3.变量和其使用尽量放到一起
- 4.【必须】 import过 长，要放在多行的时候，使用 from xxx import（a, b, c）,不要用 \ 换行
- 5.Django Model 定义的 choices 直接在定义在类里面

- 6.【必须】参考蓝鲸应用开发框架，把 Models 的初始化代码放到 migrations 里面

前端编码规范

命名规则

类的基础命名方式：“项目英文简写-当前页-页面内容块”如 .ijobs-index-box。

Id 的基础命名方式：语义化，并使用下滑杠连接，如步骤名称可命名为 #step_name

Javascript 变量命名方式：按照变量类型的首个字母为前缀，使用驼峰命名法；

类型	变量名	前缀
Array	数组	a
Boolean	布尔	b
Float	浮点	l
Function	函数	f
Integer(int)	整型	n
Object	对象	o
Regular Expression	正则	r
String	字符串	s

例子：

```
var aName = ['zhangsan','lizi','zhaowu'];  //Array 数组
var oBtn = window.document.getElementById('btn');  //Object 对象
function fnName(){};  //Function 函数
var nAge = 25;  //Integer(int) 整型
var sWebURL="www.wangyingran.com";  //String 字符串
```

日志规范

【必须】日志输出级别

- Error：系统异常，影响用户使用，必须通知到开发者及时修复。
- Warning：系统异常，不影响用户使用，但有异常需要跟进修复。
- Info：系统流水日志或日常记录。不做通知配置

日志输出格式

- 日志需要包含当前函数名，方便查找和定位

- 重要操作流水直接记录数据库，可以保存较长时间
- 错误日志要方便定位问题，建议记录当前参数以及上下文变量

【必须】代码提交规范

每次代码提交必须有备注说明，注明本次提交做了哪些修改

commit 分类：

- 1.bugfix: ——— 线上功能 bug
- 2.sprintfix: —— 未上线代码修改 （ 功能模块未上线部分bug ）
- 3.minor: ———— 不重要的修改（ 换行，拼写错误等 ）
- 4.feature: —— 新功能说明

接口规范

【必须】Api 的 method

Api 的 method，要根据实际请求的类型，查询一定要用 get，不能随意指定 method。具体可以参考文档 《RESTful Web Services Cookbook-cn.pdf》 。

【必须】接口返回内容

接口返回内容开发建议直接参考蓝鲸 apigateway 规范，返回的内容中包含 result code data message request_id 这几个字段

字段名	返回内容描述
result	True/False
code	现阶段可以不使用, 0代表正确，非0 代表不同的错误情况；
data	成功时，返回的数据的内容
message	失败时，返回的错误信息
request_id 标识	请求的id（可以自动生成的唯一标识，方便追踪请求记录 uuid）

接口返回Status Code

建议充分利用 HTTP Status Code 作为响应结果的基本状态码，基本状态码不能区分的 status，再用响应 body 中”约定”的 code 进行补充

http状态码详细说明请参考： https://zh.wikipedia.org/wiki/HTTP%E7%8A%B6%E6%80%81%E7%A0%81

接口数据验证

数据检验逻辑”应当”和业务逻辑分离，这样做的好处：

- 1.代码逻辑更加清晰
- 2.数据校验逻辑（可能）可以复用

  分离方案：

  1）form-data/url-params：Django-Form

  2）Json：Json-schema

代码注释

【必须】Python 代码注释

- 1.方法必须使用标注注释，如果是公有方法或对外提供的 API 相关方法，则最好给出使用样例。如：

```
def render_mako(template_name, dictionary={}, context_instance=None):
    '''
    render the mako template and return the HttpResponse

    @param template_name: 模板名字
    @param dictionary: context字典
    @param context_instance: 初始化context，如果要使用 TEMPLATE_CONTEXT_PROCESSORS，则要使用RequestContext(request)
    @note: 因为返回是HttpResponse，所以这个方法也适合给ajax用(dataType不是json的ajax)
    @Example:
                render_mako('mako_temp.html',{'form':form})
            or
                render_mako('mako_temp.html',{'form':form},RequestContext(request))
            or
                render_mako('mako_temp.html',{},RequestContext(request, {'form':form}))
    '''

    mako_temp = mylookup.get_template(template_name)
```

- 2.Module注释：在开头要加入对该模块的注释。如：

```python
# -*- coding: utf-8 -*-
'''
@summary: 始化logger实例(对logging的封装)
@usage:
          >>> from common.log import logger
          >>> logger.error(u'系统开小差了！')
'''

# 使用python的logging模块，配合settings的LOGGING属性
import logging
import sys
import os
import traceback
```

- 3.普通代码注释应该以‘#’和单个空格开始。
- 4.如果有需要调整或者需要优化的地方，可以使用 #TODO 这里是注释内容”进行注释，格式：'#'+单个空格+'TODO'+单个空格+注释内容。
- 5.方法的返回，如果数据结构比较复杂，则必须要对返回结果的每个属性做解释。

【必须】前端代码注释

Html 注释

<!-- 容器 -->

...
<!-- 用户名 -->

...

<!-- /用户名 -->
...

<!-- /容器 -->

css 注释

内容比较少是可以只在顶部加注释，内容比较多时在尾部加结束标签/* 注释内容 end */

/* 新建任务 start */
.new-task{}
/* 新建任务名 */

```
.task-name{color:#333;}
/* 新建任务时间 */
.task-created-time{background:url(img/clock.png) no-repeat;}
/* 新建任务 end */
```

Js 注释

Js注释同上，函数如果有参数，建议简单备注一下参数的内容和类型。

Js 函数

更多规范请参考： http://magicbox.bk.tencent.com/#doc/show?id=html_structure

文件或包的引用

- 1.Python 代码：模块或包的引入最好使用完整路径，即使是同一个包下的相互引用，也建议使用完整路径。这样比较方便代码阅读，同时若后续需修改为相对路径也很简单。
- 2.【必须】前端页面：在页面中引用 css 和 js、或配置 url 路径时，必须使用“绝对路径”，而不要使用‘../’，‘./’等相对路径的引用方式