**odoo V10中文参考手册（指导规范）**

[鼎易软件](https://www.jianshu.com/u/6fdae8ec06bc)

2017.04.13

# 指导规范

## 模块构造

### 文件夹

模块的文件夹列表及对应作用：

* data/ 演示和实际数据的xml
* models/ 模型定义
* controllers/ 包含控制器
* views/ 包含视图和模板
* static/ 包含页面相关的，一般划分为css/,/js,/img...  
  其他可选的文件夹：
* wizard/ 放临时的model和视图
* report/ 存放报表相关的python对象和xml
* tests/ 存放python和yml测试用例

### 文件命名

一般把后台视图和前端页面视图分两个文件夹存放。一系列业务model放置在一个文件里，如果只有一个model它的名字就与模型名一致。如：

* models/<main\_model>.py
* models/<inherited\_main\_model>.py
* views/<main\_model>\_templates.xml
* views/<main\_model>\_views.xml

数据文件根据其内容分开命名，如果demo和data，文件名是以主模型名命令，并以\_demo.xml,\_data.xml结尾  
控制器中唯一的文件被命名为main.py。如果需要从其他的模块中继承控制器，就命令为<module\_name>.py  
由于静态文件在前端页面和后台页面是共用的，css、js、xml文件一般以该组的名字来结尾如：im\_chat\_common.css, im\_chat\_common.js,如果模块只有唯一一个文件，就直接用module\_name.ext  
对于wizards，一般命名为<main\_transient>.py,<main\_transient>\_views.xml  
对于报表，一般命名为<report\_name\_A>\_report.py,<report\_name\_A>\_report\_views.py  
对于打印报表，一般命名为<print\_report\_name>\_reports.py , <print\_report\_name>\_templates.xml

完整的文件列表如下：

addons/<my\_module\_name>/

|-- \_\_init\_\_.py

|-- \_\_manifest\_\_.py

|-- controllers/

| |-- \_\_init\_\_.py

| |-- <inherited\_module\_name>.py

| `-- main.py

|-- data/

| |-- <main\_model>\_data.xml

| `-- <inherited\_main\_model>\_demo.xml

|-- models/

| |-- \_\_init\_\_.py

| |-- <main\_model>.py

| `-- <inherited\_main\_model>.py

|-- report/

| |-- \_\_init\_\_.py

| |-- <main\_stat\_report\_model>.py

| |-- <main\_stat\_report\_model>\_views.xml

| |-- <main\_print\_report>\_reports.xml

| `-- <main\_print\_report>\_templates.xml

|-- security/

| |-- ir.model.access.csv

| `-- <main\_model>\_security.xml

|-- static/

| |-- img/

| | |-- my\_little\_kitten.png

| | `-- troll.jpg

| |-- lib/

| | `-- external\_lib/

| `-- src/

| |-- js/

| | `-- <my\_module\_name>.js

| |-- css/

| | `-- <my\_module\_name>.css

| |-- less/

| | `-- <my\_module\_name>.less

| `-- xml/

| `-- <my\_module\_name>.xml

|-- views/

| |-- <main\_model>\_templates.xml

| |-- <main\_model>\_views.xml

| |-- <inherited\_main\_model>\_templates.xml

| `-- <inherited\_main\_model>\_views.xml

`-- wizard/

|-- <main\_transient\_A>.py

|-- <main\_transient\_A>\_views.xml

|-- <main\_transient\_B>.py

`-- <main\_transient\_B>\_views.xml

## XML文件

### 格式

当定义一个记录的xml时，需要一个<record>标记:

* 将id属性放在model前面
* 对于字段的字义，name属性放在最前面，然后放value,eval,之后再按重要程度放其他属性如widget, options
* 根据model将record进行分组，但在action/menu/views有依赖关系时该规则不好用
* 使用好的命名习惯
* <data>标签只在设置不可更新的数据时用，使用noupdate=1

<record id="view\_id" model="ir.ui.view">

<field name="name">view.name</field>

<field name="model">object\_name</field>

<field name="priority" eval="16"/>

<field name="arch" type="xml">

<tree>

<field name="my\_field\_1"/>

<field name="my\_field\_2" string="My Label" widget="statusbar" statusbar\_visible="draft,sent,progress,done" />

</tree>

</field>

</record>

odoo还支持一些自定义的标签如：

* menuitem - 用于定义一个ir.ui.menu
* workflow - <workflow>标签向工作流发送信号
* template - 用于定义一个只需要arch片段的view
* report - 用于定义报表action
* act\_window- 当record用不了的时候用它

### 命名xml\_id

#### 安全、视图和action

* 菜单 - <model\_name>\_menu
* 视图 - <model\_name>\_view\_<view\_type>, view\_type=>kanban, form, tree, search...
* action - 主action命名为<model\_name>\_action,其他用\_<detail>后缀，detail可用于简要描述该action功能
* 分组 - <model\_name>\_group\_<group\_name> ,group\_name是分组名如用户、管理员……
* 规则 - <model\_name>\_rule\_<concerned\_group>,concerned\_group 代表对应组的简写如user，public...

<!-- views and menus -->

<record id="model\_name\_view\_form" model="ir.ui.view">

...

</record>

<record id="model\_name\_view\_kanban" model="ir.ui.view">

...

</record>

<menuitem

id="model\_name\_menu\_root"

name="Main Menu"

sequence="5"

/>

<menuitem

id="model\_name\_menu\_action"

name="Sub Menu 1"

parent="module\_name.module\_name\_menu\_root"

action="model\_name\_action"

sequence="10"

/>

<!-- actions -->

<record id="model\_name\_action" model="ir.actions.act\_window">

...

</record>

<record id="model\_name\_action\_child\_list" model="ir.actions.act\_window">

...

</record>

<!-- security -->

<record id="module\_name\_group\_user" model="res.groups">

...

</record>

<record id="model\_name\_rule\_public" model="ir.rule">

...

</record>

<record id="model\_name\_rule\_company" model="ir.rule">

...

</record>

#### 继承xml

继承视图的命名方式是<base\_view>\_inherit\_<current\_module\_name>

<record id="inherited\_model\_view\_form\_inherit\_my\_module" model="ir.ui.view">

...

</record>

## python

### PEP8

使用linter来帮助显示语法错误和警告，odoo遵循了大部分的python标准，但忽略了以下几项：

* E501: line too long
* E301: expected 1 blank line, found 0
* E302: expected 2 blank lines, found 1
* E126: continuation line over-indented for hanging indent
* E123: closing bracket does not match indentation of opening bracket's line
* E127: continuation line over-indented for visual indent
* E128: continuation line under-indented for visual indent
* E265: block comment should start with '# '

### 导入

导入的顺序为：

1. 导入标准库，每个库一行
2. 导入odoo
3. 从odoo模块导入（需要的时候）

# 1 : imports of python lib

import base64

import re

import time

from datetime import datetime

# 2 : imports of odoo

import odoo

from odoo import api, fields, models # alphabetically ordered

from odoo.tools.safe\_eval import safe\_eval as eval

from odoo.tools.translate import \_

# 3 : imports from odoo modules

from odoo.addons.website.models.website import slug

from odoo.addons.web.controllers.main import login\_redirect

### 符合习惯的python编程

* 每个文件放置# -\*- coding: utf-8 -\*-在头一行
* 从更易读的角度出发
* 不用clone()

# bad

new\_dict = my\_dict.clone()

new\_list = old\_list.clone()

# good

new\_dict = dict(my\_dict)

new\_list = list(old\_list)

* Python字典：创建和修改

# -- creation empty dict

my\_dict = {}

my\_dict2 = dict()

# -- creation with values

# bad

my\_dict = {}

my\_dict['foo'] = 3

my\_dict['bar'] = 4

# good

my\_dict = {'foo': 3, 'bar': 4}

# -- update dict

# bad

my\_dict['foo'] = 3

my\_dict['bar'] = 4

my\_dict['baz'] = 5

# good

my\_dict.update(foo=3, bar=4, baz=5)

my\_dict = dict(my\_dict, \*\*my\_dict2)

* 使用有意义的变量名、函数名、类名
* 临时变量有时可以让赋值给对象更简单：

# pointless

schema = kw['schema']

params = {'schema': schema}

# simpler

params = {'schema': kw['schema']}

* 为了简单可使用多重返回

# a bit complex and with a redundant temp variable

def axes(self, axis):

axes = []

if type(axis) == type([]):

axes.extend(axis)

else:

axes.append(axis)

return axes

# clearer

def axes(self, axis):

if type(axis) == type([]):

return list(axis) # clone the axis

else:

return [axis] # single-element list

* 内置函数
* 正确使用list,dict,map
* 集合也可以当成布尔型

bool([]) is False

bool([1]) is True

bool([False]) is True

* 循环

# creates a temporary list and looks bar

for key in my\_dict.keys():

"do something..."

# better

for key in my\_dict:

"do something..."

# creates a temporary list

for key, value in my\_dict.items():

"do something..."

# only iterates

for key, value in my\_dict.iteritems():

"do something..."

* 使用dict.setdefault

# longer.. harder to read

values = {}

for element in iterable:

if element not in values:

values[element] = []

values[element].append(other\_value)

# better.. use dict.setdefault method

values = {}

for element in iterable:

values.setdefault(element, []).append(other\_value)

* 添加注释

### odoo中编程

* 避免自定义生成器和装饰器，只使用odoo API已有的
* 使用更易理解的方法名

#### 让你的方法可以批量处理

当添加一个函数时，确保它可以处理多重数据，如通过api.multi()装饰器，可以在self上进行循环处理

@api.multi

def my\_method(self)

for record in self:

record.do\_cool\_stuff()

避免使用api.one装饰器，因为它可能不会像你想象中一样工作。  
为了更好的性能，比如当定义一个状态按钮时，不在api.multi循环里用search和search\_count方法，而用read\_group一次计算

@api.multi

def \_compute\_equipment\_count(self):

""" Count the number of equipement per category """

equipment\_data = self.env['hr.equipment'].read\_group([('category\_id', 'in', self.ids)], ['category\_id'], ['category\_id'])

mapped\_data = dict([(m['category\_id'][0], m['category\_id\_count']) for m in equipment\_data])

for category in self:

category.equipment\_count = mapped\_data.get(category.id, 0)

#### 扩散上下文环境

在新API中，context变量是不能修改的。可以通过with\_context来使用新的运行环境调用方法。

records.with\_context(new\_context).do\_stuff() # all the context is replaced

records.with\_context(\*\*additionnal\_context).do\_other\_stuff() # additionnal\_context values override native context ones

如果需要创建一个新的context来对某对象进行操作， 选择一个好名字，以模块名为前缀用隔离它的影响，如：mail\_create\_nosubscribe, mail\_notrack, mail\_notify\_user\_signature...

#### 尽量使用ORM

当ORM可以实现的时候尽量使用ORM而不要直接写sql，因为它可能会绕过orm的一些规则如权限、事务等  
，还会让代码变得难读且不安全。

# very very wrong

self.env.cr.execute('SELECT id FROM auction\_lots WHERE auction\_id in (' + ','.join(map(str, ids))+') AND state=%s AND obj\_price > 0', ('draft',))

auction\_lots\_ids = [x[0] for x in self.env.cr.fetchall()]

# no injection, but still wrong

self.env.cr.execute('SELECT id FROM auction\_lots WHERE auction\_id in %s '\

'AND state=%s AND obj\_price > 0', (tuple(ids), 'draft',))

auction\_lots\_ids = [x[0] for x in self.env.cr.fetchall()]

# better

auction\_lots\_ids = self.search([('auction\_id','in',ids), ('state','=','draft'), ('obj\_price','>',0)])

#### 不要进行sql注入

不要用python的+号连接符、%解释符来拼sql

self.env.cr.execute('SELECT distinct child\_id FROM account\_account\_consol\_rel ' +

'WHERE parent\_id IN ('+','.join(map(str, ids))+')')

# better

self.env.cr.execute('SELECT DISTINCT child\_id '\

'FROM account\_account\_consol\_rel '\

'WHERE parent\_id IN %s',

(tuple(ids),))

#### 尽量把方法写的短而简单

#### 不要手动提交事务

odoo有自己的一套机制用于事务处理

#### 正确的使用翻译方法

odoo用一个下划线方法来表明某字段需要翻译，该方法通过from odoo.tools.translate import \_导入  
一般情况下该方法只能被用在手动写在代码里的字符串的翻译，不能用在动态的值中，翻译方法的调用只能是\_('literal string')，里面不能加其他的。

# good: plain strings

error = \_('This record is locked!')

# good: strings with formatting patterns included

error = \_('Record %s cannot be modified!') % record

# ok too: multi-line literal strings

error = \_("""This is a bad multiline example

about record %s!""") % record

error = \_('Record %s cannot be modified' \

'after being validated!') % record

# bad: tries to translate after string formatting

# (pay attention to brackets!)

# This does NOT work and messes up the translations!

error = \_('Record %s cannot be modified!' % record)

# bad: dynamic string, string concatenation, etc are forbidden!

# This does NOT work and messes up the translations!

error = \_("'" + que\_rec['question'] + "' \n")

# bad: field values are automatically translated by the framework

# This is useless and will not work the way you think:

error = \_("Product %s is out of stock!") % \_(product.name)

# and the following will of course not work as already explained:

error = \_("Product %s is out of stock!" % product.name)

# bad: field values are automatically translated by the framework

# This is useless and will not work the way you think:

error = \_("Product %s is not available!") % \_(product.name)

# and the following will of course not work as already explained:

error = \_("Product %s is not available!" % product.name)

# Instead you can do the following and everything will be translated,

# including the product name if its field definition has the

# translate flag properly set:

error = \_("Product %s is not available!") % product.name

格式化字符串如%s, %d需要被保留，并以更明确的方式使用：

# Bad: makes the translations hard to work with

error = "'" + question + \_("' \nPlease enter an integer value ")

# Better (pay attention to position of the brackets too!)

error = \_("Answer to question %s is not valid.\n" \

"Please enter an integer value.") % question

### 符号和习惯

* 模型名-使用.分隔，模块名做前缀
* 定义odoo模型时，使用单数形式的名字如res.user,res.partner
* 定义wizard时，使用<related\_base\_model>.<action>，related\_base\_model 代表其所相关的模型名，action是其功能简称，如account.invoice.make
* 定义报表模型时，使用<related\_base\_model>.report.<action>，和wizard一样
* python类-使用驼峰命名方式

class AccountInvoice(models.Model):

...

class account\_invoice(osv.osv):

...

* 变量名
* 模型变量使用驼峰命名方式
* 普通变量用下划线+小写字母
* 由于新api中记录是集合形式，当变量不包含id时不以id作后缀

ResPartner = self.env['res.partner']

partners = ResPartner.browse(ids)

partner\_id = partners[0].id

One2Many， Many2Many字段一般以ids作为后缀如：sale\_order\_line\_ids  
Many2One 一般以\_id为后缀如：partner\_id, user\_id

* 方法习惯
* 计算字段- 计算方法一般是\_compute\_<field\_name>
* 搜索方法- \_search\_<field\_name>
* 默认方法- \_default\_<field\_name>
* onchange方法- \_onchange\_<field\_name>
* 约束方法 - \_check\_<constraint\_name>
* action方法- 一个对象的动作方法一般以action\_开头，它的装饰器是@api.multi，如果它只使用单条计算，可在方法头添加self.ensure\_one()
* 模型中属性的顺序
* 私有属性：\_name, \_description, \_inherit, ...
* 默认方法和\_default\_get
* 字段声明
* 计算和搜索方法和字段声明顺序一致
* 约束方法(@api.constrains)和onchange方法(@api.onchange)
* CRUD方法
* action方法
* 其他业务方法

class Event(models.Model):

# Private attributes

\_name = 'event.event'

\_description = 'Event'

# Default methods

def \_default\_name(self):

...

# Fields declaration

name = fields.Char(string='Name', default=\_default\_name)

seats\_reserved = fields.Integer(oldname='register\_current', string='Reserved Seats',

store=True, readonly=True, compute='\_compute\_seats')

seats\_available = fields.Integer(oldname='register\_avail', string='Available Seats',

store=True, readonly=True, compute='\_compute\_seats')

price = fields.Integer(string='Price')

# compute and search fields, in the same order of fields declaration

@api.multi

@api.depends('seats\_max', 'registration\_ids.state', 'registration\_ids.nb\_register')

def \_compute\_seats(self):

...

# Constraints and onchanges

@api.constrains('seats\_max', 'seats\_available')

def \_check\_seats\_limit(self):

...

@api.onchange('date\_begin')

def \_onchange\_date\_begin(self):

...

# CRUD methods (and name\_get, name\_search, ...) overrides

def create(self, values):

...

# Action methods

@api.multi

def action\_validate(self):

self.ensure\_one()

...

# Business methods

def mail\_user\_confirm(self):

...

## Javascript和CSS

javascript:

* 在所有javascript文件中使用use strict;
* 使用linter
* 不添加压缩javascript库
* 类名使用驼峰命名
* 如果javascript代码需要全局运行，在website模块中声明一个if\_dom\_contains 方法

odoo.website.if\_dom\_contains('.jquery\_class\_selector', function () {

/\*your code here\*/

});

CSS:

* 将所有的class命名为o\_<module\_name>,如o\_forum
* 避免使用id
* 使用bootstrap的class
* 用下划线+小写来命名

## GIT

建议看专门的GIT教程

# 更新API

http://www.odoo.com/documentation/10.0/reference/upgrade\_api.html

译自odoo官方文档：http://www.odoo.com/documentation/10.0/reference/guidelines.html ，不当之处欢迎批评指正。

内容发布自[http://www.jianshu.com/u/6fdae8ec06bc](https://www.jianshu.com/u/6fdae8ec06bc)，转载请注明出处