odoo销售流程代码简单剖析

2016年06月06日

1、新建销售报价单

2、点击确认订单，将报价单转为销售订单，同时生成对应出库单，相关代码：

确定按钮执行的方法为 sale.order 的 action\_button\_confirm 方法，具体代码：

**def action\_button\_confirm**(self, cr, uid, ids, context=None):

**if not** context:

context = {}

**assert** len(ids) == 1, 'This option should only be used for a single id at a time.'

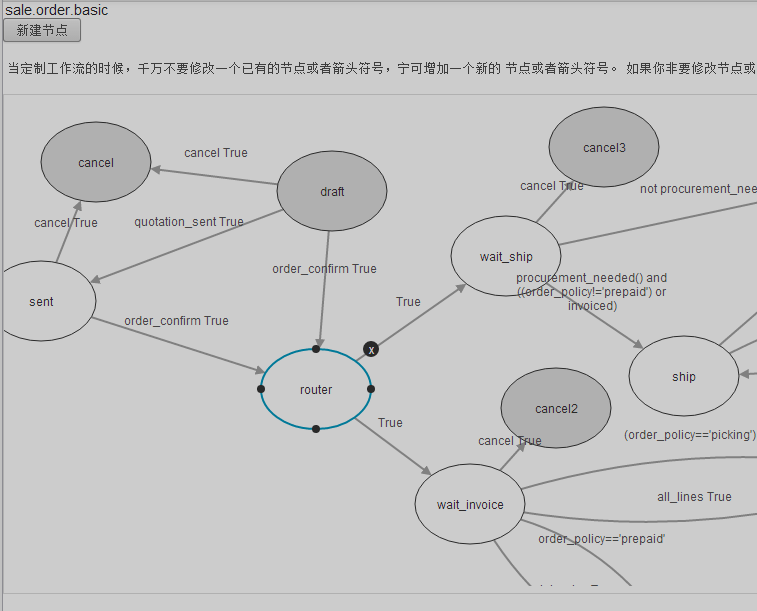
self.signal\_workflow(cr, uid, ids, 'order\_confirm')

**if** context.get('send\_email'):

self.force\_quotation\_send(cr, uid, ids, context=context)

**return** True

可以看到代码有触发工作流，来确认销售订单；借住工作流视图



我们是从 draft 执行 order\_confirm 进入router，此处我们打开看到，执行了 sale.order 类的 action\_wait 方法，具体代码：

**def action\_wait**(self, cr, uid, ids, context=None):#只处理自身的销售订单和产品清单的信息更新

context = context **or** {}

**for** o **in** self.browse(cr, uid, ids):

**if not** any(line.state != 'cancel' **for** line **in** o.order\_line):

**raise** osv.except\_osv(\_('Error!'),\_('You cannot confirm a sales order which has no line.'))

noprod = self.test\_no\_product(cr, uid, o, context)

**if** (o.order\_policy == 'manual') **or** noprod:

self.write(cr, uid, [o.id], {'state': 'manual', 'date\_confirm': fields.date.context\_today(self, cr, uid, context=context)})

**else**:

self.write(cr, uid, [o.id], {'state': 'progress', 'date\_confirm': fields.date.context\_today(self, cr, uid, context=context)})

self.pool.get('sale.order.line').button\_confirm(cr, uid, [x.id **for** x **in** o.order\_line **if** x.state != 'cancel'])

**return** True

代码逻辑很清晰，不做进一步分析，进入 router 之后，发票部分我们暂时忽略；我们来看 wait\_ship 部分，通过 sale.order 类的 procurement\_needed 方法的返回值来确认程序的继续走向，具体代码：

**def procurement\_needed**(self, cr, uid, ids, context=None):

#when sale is installed only, there is no need to create procurements, that's only

#further installed modules (sale\_service, sale\_stock) that will change this.

sale\_line\_obj = self.pool.get('sale.order.line')

res = []

**for** order **in** self.browse(cr, uid, ids, context=context):

res.append(sale\_line\_obj.need\_procurement(cr, uid, [line.id **for** line **in** order.order\_line **if** line.state != 'cancel'], context=context))

**return** any(res)

作者在测试的时候，上面的方法的返回值是True，工作流走到了 ship ，此处工作流执行的方法是 sale.order 类的  action\_ship\_create 方法，代码：

**def action\_ship\_create**(self, cr, uid, ids, context=None):

*"""Create the required procurements to supply sales order lines, also connecting*

*the procurements to appropriate stock moves in order to bring the goods to the*

*sales order's requested location.*

***:return****: True*

*"""*

context = context **or** {}

context['lang'] = self.pool['res.users'].browse(cr, uid, uid).lang

procurement\_obj = self.pool.get('procurement.order')

sale\_line\_obj = self.pool.get('sale.order.line')

**for** order **in** self.browse(cr, uid, ids, context=context):

proc\_ids = []

vals = self.\_prepare\_procurement\_group(cr, uid, order, context=context)

**if not** order.procurement\_group\_id:

group\_id = self.pool.get("procurement.group").create(cr, uid, vals, context=context)

order.write({'procurement\_group\_id': group\_id})

**for** line **in** order.order\_line:

**if** line.state == 'cancel':

**continue**

#Try to fix exception procurement (possible when after a shipping exception the user choose to recreate)

**if** line.procurement\_ids:

#first check them to see if they are in exception or not (one of the related moves is cancelled)

procurement\_obj.check(cr, uid, [x.id **for** x **in** line.procurement\_ids **if** x.state **not in** ['cancel', 'done']])

line.refresh()

#run again procurement that are in exception in order to trigger another move

except\_proc\_ids = [x.id **for** x **in** line.procurement\_ids **if** x.state **in** ('exception', 'cancel')]

procurement\_obj.reset\_to\_confirmed(cr, uid, except\_proc\_ids, context=context)

proc\_ids += except\_proc\_ids

**elif** sale\_line\_obj.need\_procurement(cr, uid, [line.id], context=context):

**if** (line.state == 'done') **or not** line.product\_id:

**continue**

vals = self.\_prepare\_order\_line\_procurement(cr, uid, order, line, group\_id=order.procurement\_group\_id.id, context=context)

ctx = context.copy()

ctx['procurement\_autorun\_defer'] = True

proc\_id = procurement\_obj.create(cr, uid, vals, context=ctx)#此处执行了procurement\_order表记录的生成，

proc\_ids.append(proc\_id)

#Confirm procurement order such that rules will be applied on it

#note that the workflow normally ensure proc\_ids isn't an empty list

procurement\_obj.run(cr, uid, proc\_ids, context=context)

#此处run执行了stock\_move表记录的生成等一系列操作，执行的run顺序===》addons/stock/procurement.py（line 208）

#===》addons/procurement/procurement.py（line 197）,stock（前者）扩展了pro（后者）的run方法

#if shipping was in exception and the user choose to recreate the delivery order, write the new status of SO

**if** order.state == 'shipping\_except':

val = {'state': 'progress', 'shipped': False}

**if** (order.order\_policy == 'manual'):

**for** line **in** order.order\_line:

**if** (**not** line.invoiced) **and** (line.state **not in** ('cancel', 'draft')):

val['state'] = 'manual'

**break**

order.write(val)

**return** True

上面代码生成与销售订单对应的补货记录，同时生成与销售订单对应的出库单，**注意**这里，在追代码的时候，procurement\_obj.run 处调用，代码里只能看到相应的stock.move记录的生成，并没有对应出库单stock.picking的生成，一下代码是procurement\_obj.run 调用的代码追踪：

**def run**(self, cr, uid, ids, autocommit=False, context=None):

new\_ids = [x.id **for** x **in** self.browse(cr, uid, ids, context=context) **if** x.state **not in** ('running', 'done', 'cancel')]

res = super(procurement\_order, self).run(cr, uid, new\_ids, autocommit=autocommit, context=context)

#after all the procurements are run, check if some created a draft stock move that needs to be confirmed

#(we do that in batch because it fasts the picking assignation and the picking state computation)

move\_to\_confirm\_ids = []

**for** procurement **in** self.browse(cr, uid, new\_ids, context=context):

**if** procurement.state == "running" **and** procurement.rule\_id **and** procurement.rule\_id.action == "move":

move\_to\_confirm\_ids += [m.id **for** m **in** procurement.move\_ids **if** m.state == 'draft']

**if** move\_to\_confirm\_ids:

self.pool.get('stock.move').action\_confirm(cr, uid, move\_to\_confirm\_ids, context=context)

**return** res

上面为stock模块扩展的run方法，super部分调用到父类 run 方法, 此处代码标记为1代

**def run**(self, cr, uid, ids, autocommit=False, context=None):#主要是变更procurement\_order表的信息

**for** procurement\_id **in** ids:

#we intentionnaly do the browse under the for loop to avoid caching all ids which would be resource greedy

#and useless as we'll make a refresh later that will invalidate all the cache (and thus the next iteration

#will fetch all the ids again)

procurement = self.browse(cr, uid, procurement\_id, context=context)

**if** procurement.state **not in** ("running", "done"):

**try**:

**if** self.\_assign(cr, uid, procurement, context=context):

res = self.\_run(cr, uid, procurement, context=context **or** {})

#上一条 \_run先调用了addons/stock/procurement.py（line 196）\_run方法，生成stock\_move记录

**if** res:

self.write(cr, uid, [procurement.id], {'state': 'running'}, context=context)

**else**:

self.write(cr, uid, [procurement.id], {'state': 'exception'}, context=context)

**else**:

self.message\_post(cr, uid, [procurement.id], body=\_('No rule matching this procurement'), context=context)

self.write(cr, uid, [procurement.id], {'state': 'exception'}, context=context)

**if** autocommit:

cr.commit()

**except** OperationalError:

**if** autocommit:

cr.rollback()

**continue**

**else**:

**raise**

**return** True

上面代码备注的中文部分，\_run 方法代码，调用到stock模块中扩展的\_run(下面代码)，上面代码标记为2代

**def \_run**(self, cr, uid, procurement, context=None):

#生成stock\_move表记录

**if** procurement.rule\_id **and** procurement.rule\_id.action == 'move':

**if not** procurement.rule\_id.location\_src\_id:

self.message\_post(cr, uid, [procurement.id], body=\_('No source location defined!'), context=context)

**return** False

move\_obj = self.pool.get('stock.move')

move\_dict = self.\_run\_move\_create(cr, uid, procurement, context=context)

#create the move as SUPERUSER because the current user may not have the rights to do it (mto product launched by a sale for example)

move\_obj.create(cr, SUPERUSER\_ID, move\_dict, context=context)

**return** True

**return** super(procurement\_order, self).\_run(cr, uid, procurement, context=context)

上面代码编辑为3代，流程分析：1代中res = super 跳入2代里执行，2代res = self.\_run（具体执行代码在3代里）部分执行的结果生成销售订单对应的stock.move记录，然后返回到2代再返回到1代，1代里继续执行，此处即是**注意点**，注意代码self.pool.get('stock.move').action\_confirm，此处代码跳转到stock.move类中，picking 的生成即在stock.move 的 confirm 方法中实现的，剩下代码不做进一步详解，如有兴趣，可留言与我一起讨论分析

**\*以上分析为本人在学习odoo过程中的一些学习记录，如有错误请斧正，以免误导其他odoo学者**