



# OpenERP技术培训

出品单位: OpenERP中文社区



## ■ OpenERP简介:

OpenERP是一款基于Python 语言的开源企业管理软件。它是跨平台的，而且，同时支持C/S 和B/S 两种架构形式，另外，OpenERP的模块化架构非常著称。官方标准发布版中有150个模块，涵盖财务、ERP、CRM、项目管理等常用功能。由开源社区提供的模块超过1000个，涵盖各行业、各领域的企业管理功能。

## ■ OpenERP中文社区简介:

OpenERP中文社区，从2007 年开始介绍OpenERP 到中国，主要论坛在这里：<http://www.shine-it.net/>，论坛的QQ群是：69195329。在中文社区的努力下，目前完成了OpenERP的汉化及中文文档编译等工作。另外，提供有偿咨询、实施、开发等服务，在这里：<http://openerp-china.org/>。

## 关于作者

### ■ 主编：老肖

上海-老肖(QQ: 1417063315)，老肖致力于OpenERP在中国的普及工作。编写过《OpenERP应用和开发基础》等书籍。为多家单位成功开发、实施过OpenERP。

### ■ 编辑

青岛-芹菜(630682763)、南京-海飞(330472962)、深圳-浪打浪(41473064)、四川♂Toop(139776)等

### ■ 审稿

上海-digitalsatori(309401007)、上海-Jeff(85822082)

# 培训目标

## ■ 学习安装和管理

- 模块使用
- 模块安装
- 配置和管理

## ■ 学会新模块开发

- 对象(Object)
- 视图(View)
- 工作流(Workflow)
- 向导(Wizard)
- 报表(Report)

## ■ 学会集成接口开发

# 培训计划 – 第1天

## ■ OpenERP简介

## ■ LaunchPad ( 取得源代码! )

## ■ 安装

- 客户端(Client), 服务端(Serveur)
- 数据库(PostgreSQL)

## ■ 接口(Interface)

## ■ 配置和管理(Administration)

- 语言包安装
- 用户和权限组(User and Group)
- 权限控制(ACL)
- 模块(Module)
- 备份(Backup)

- 架构
- 一切都是对象
  - 深入对象内部
- 介绍对象(Object, or Resource, or Model)
  - 如何定义简单的对象
- 格式化XML
  - 语法和用法
- 介绍视图(View)
  - 原理和语法

### ■ 视图详解

- 高级标签
- 继承

### ■ 工作流(Workflow)详解

- 用法
- 原理
- 定义/定制

### ■ 报表(Report)

- 原理
  - OpenReport, Tiny RML2PDF
- 注册
- 报表类型
  - RML
  - OpenOffice报表

### ■ 向导(Wizard)

- 原理
- 定义
- 注册

### ■ 链接(Action)

- 原理
- 定义



### ■ Web-Services接口：XML-RPC

- Python范例
- Ruby范例

## ■我们从哪开始呢？

## ■ OpenERP

- 版权：ERP（GNU GPL）开源协议
- Python: <http://www.python.org> 版本  $\geq 2.4$
- 客户机/服务器
- 3层架构
  - PostgreSQL
  - OpenERP - 服务器
  - OpenERP - 客户（Web-Client, GTK-Client）

# 从LaunchPad 获得源码

## ■ LaunchPad介绍

- 所有开发相关资料集中在一台服务器上
- 世界范围协作开发的门户网站
  - 源代码版本管理（以Bazaar为基础）
  - 翻译管理
  - bug管理
  - 产品特征管理（蓝图）
  - FAQ管理

■ LaunchPad网址：<http://www.launchpad.net/openobject>

# LaunchPad – Bazaar的安装

- \* 添加到更新源

```
sudo gedit /etc/apt/sources.list
```

- \* 加入这一行

```
deb http://ppa.launchpad.net/bzr/ubuntu intrepid main
```

- \* 开始安装:

```
sudo apt-get update  
sudo apt-get install bzr
```

## LaunchPad – 下载OPENERP源码

\*创建一个工作目录:

```
$> mkdir ~/openerp/stable/6.0 -p  
$> cd ~/openerp/stable/6.0
```

\*获取最新版本的源代码 :

```
$> bzr clone lp:openobject-server/6.0 server  
$> bzr clone lp:openobject-client/6.0 client  
$> bzr clone lp:openobject-addons/6.0 addons  
$> bzr clone lp:openobject-client-web/6.0 client_web
```

\* 获得额外插件:

```
$> bzr clone lp:~openerp-commiter/openobject-ddons/  
trunk-extra-addons extra-addons
```

\* 创建OPENERP插件目录的硬链接:

```
$> cd server/bin/addons  
$> ln -s ../../../../addons/* .
```

# Python >= 2.4的安装

## \* 常用模块安装

```
#> apt-get install python
#> apt-get install python-egenix-mxdatetime
#> apt-get install python-xml python-lxml
#> apt-get install python-libxml2 python-libxslt1
#> apt-get install pytz
#> apt-get install python-yaml
#> apt-get install python-mako
```

## \* 服务器组件安装

```
#> apt-get install python-psycopg2
#> apt-get install python-pydot graphviz python-pyparsing
#> apt-get install python-imaging python-reportlab python-pychart
#> apt-get install python-tz
```

## \* 客户端组件安装

```
#> apt-get install python-gtk2 python-glade2
#> apt-get install python-matplotlib python-hippocanvas
```

# PostgreSQL数据库的安装

\* 安装

```
#> apt-get install postgresql
```

\*创建PostgreSQL用户

```
#> su - postgres
```

```
$> createuser YOUR_USERNAME
```

\* 安装psql,命令行式的数据库客户端:

```
#> apt-get install postgresql-client-common
```

# 开始安装OPENERP

## \* 服务端配置并运行

```
$> cd ~/openerp/stable/6.0/server/bin  
$> ./openerp-server.py  
$> ./openerp-server.py --verbose  
$> ./openerp-server.py --debug  
$> ./openerp-server.py --update=all --database=DBNAME  
$> ./openerp-server.py --update=MODULE_NAME --  
database=DBNAME  
$> ./openerp-server.py --init --database=DBNAME
```

## \*运行gtk客户端

```
$> cd ~/openerp/stable/6.0/client/bin  
$> ./openerp-client.py  
$> ./openerp-client.py --verbose
```

注意

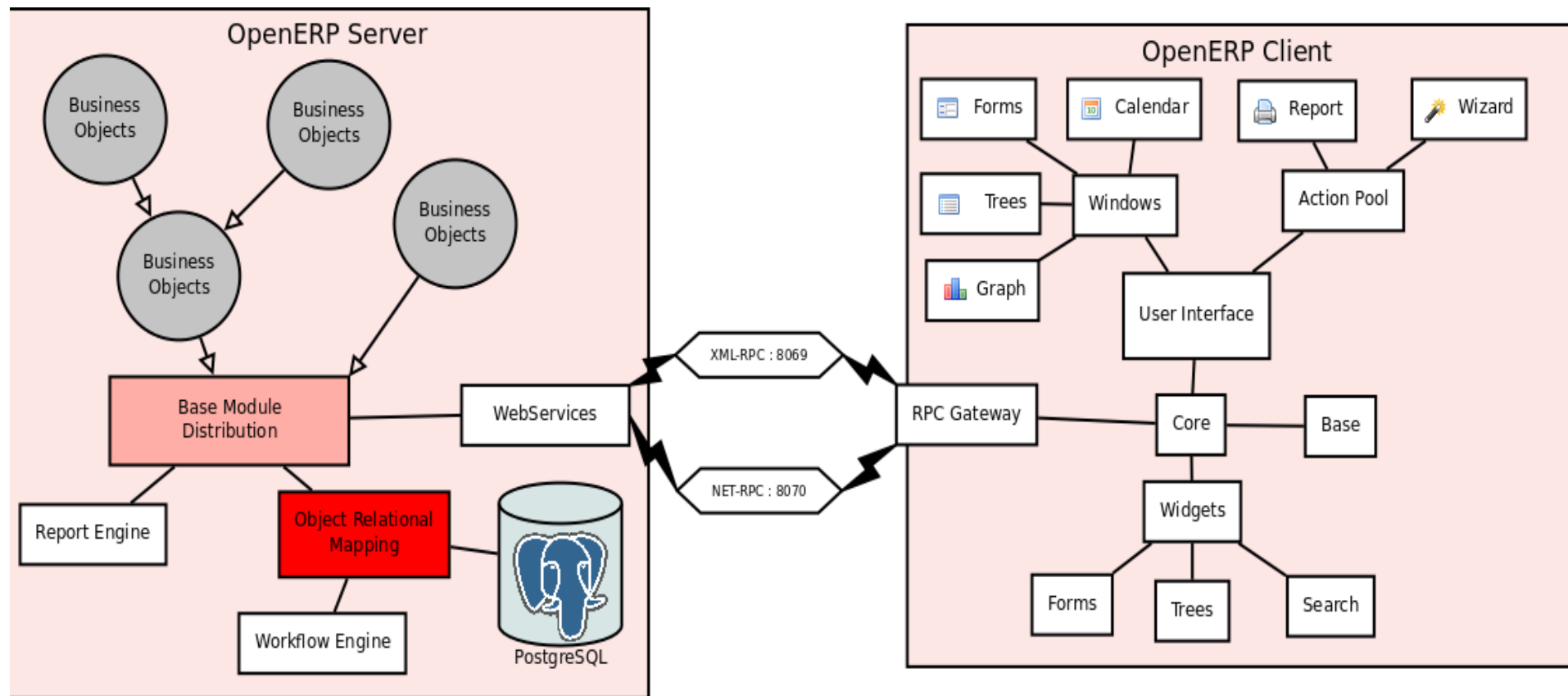
*--help* 显示帮助 !!!



# 涉及哪些技术

- 面向服务 (所有的业务逻辑都在服务器上运行)
  - 对象 (Object)
  - 对象关系映射 (ORM)
  - 视图 (View)
  - 工作流 (Workflow)
  - 向导 (Wizard)
  - 报表 (Report)
- 客户端
  - GTK Client
  - Web Client
- 通讯接口
  - XML-RPC
    - Port 8069
    - 支持服务器负载均衡
  - NET-RPC
    - Port 8070
    - 字节流, 带宽占用更小
    - 使用python对象序列化技术Pickle (python)

# 整体架构图



服务端/客户端

# 模块 **Module** (1)

- 对象(Object)
  - 字段(Columns)
  - 默认值(Default Values)
  - 约束(Constraints: constraints, sql\_constraints, check\_recursion)
  - 排序(Order)
- 视图(View)
  - Form, List, Tree, Graph, Calendar, Gantt
  - On Change
- 基础数据和演示数据(Data & Demo)
- 继承(Inherit)

## 模块 Module – 最小结构

```
module_name/  
    __init__.py  
    __openerp__.py  
    module_name.py  
    module_name_view.xml
```

### 注意:

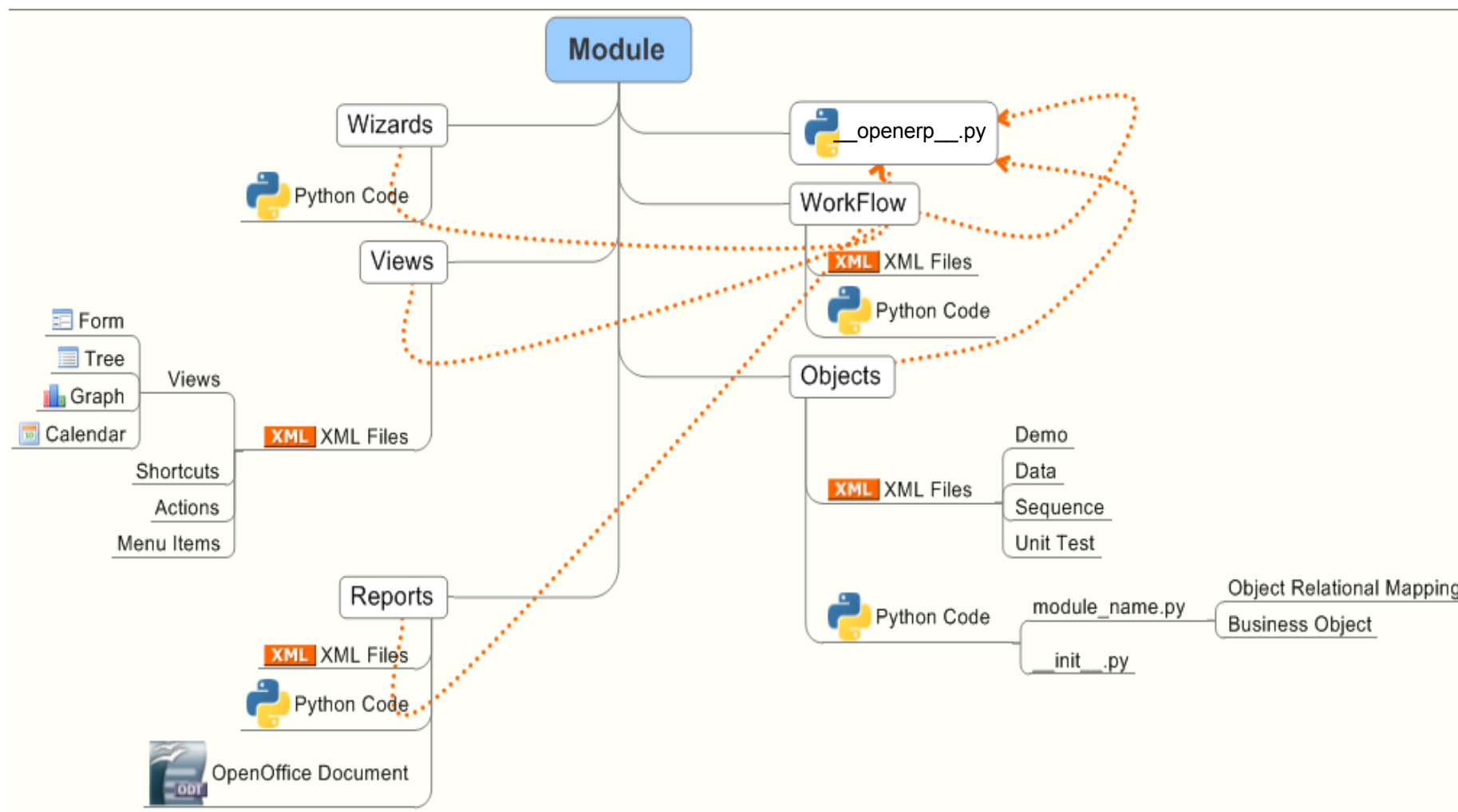
有些模块不包含module\_name.py和xml文件，  
这种特殊的模块只有模块和配置数据文件。

## 模块 Module – 完整结构

```
module_name/  
    __init__.py  
    __openerp__.py  
    module_name.py  
    module_name_2.py  
    module_name_3.py  
    module_name_view.xml  
    module_name_workflow.xml  
    module_name_wizard.xml  
    module_name_report.xml  
    module_name_demo.xml  
    module_name_data.xml  
report/  
    __init__.py  
    report.py  
wizard/  
    __init__.py  
    wizard.py
```

```
i18n/  
    fr_FR.po  
    en_US.po  
    module_name.pot  
security/  
    some.object.csv #定义权限组访问规则  
    file.xml        #定义用户组
```

# 模块 - 结构图



## \_\_init\_\_.py 和 \_\_openerp\_\_.py 简介

```
module_name / __init__.py
    import openacademy

module_name / __openerp__.py
{
    'name' : 'Module Name',
    'version' : '0.1',
    'author' : 'Tiny',
    'website' : 'http://www.openerp.com',
    'category' : 'Tools',
    'description' : 'description of this module',
    'depends' : ['base'],
    'init_xml' : ['module_name_data.xml'],
    'demo_xml' : ['module_name_demo.xml'],
    'update_xml' : [ 'module_name_view.xml',
        'module_name_workflow.xml' ],
    'installable' : True,
    'active' : False,
}
```

## module\_name / module\_name.py 说明

```
# module_name/module_name.py
from osv import osv, fields #定义需要引入的包

# OpenAcademy Training Object #解释对象的作用
class openacademy_training(osv.osv):
    _name = 'openacademy.training' #对象名称
    _description = "OpenAcademy Training" #对象描述
    _columns = { #定义字段
        'name' : fields.char('Name'),
        'date_start' : fields.date('Date' ), #开始日期
        'description' : fields.text('Description' ), #培训内容
        'active' : fields.boolean('Active' ), #本记录是否有效
    }
    _order = "date_start desc" #以开始日期升序排序
openacademy_training() #对象定义结束
```



# 模块 Module - Profile

**Profile** 模块 示例：下述**Profile**模块加载生产相关的各个**Module**，构成生产应用系统。

```
{
  'name' : 'Manufacturing industry profile',
  'version' : '1.0',
  'author' : 'OpenERP',
  'website' : 'http://www.openerp.com',
  'category' : 'Profile',
  'depends' : ['mrp', 'crm', 'sale', 'delivery'], 'demo_xml' : [],
  'update_xml' : [],
  'installable' : True,
  'active' : False,
}
```

# 对象关系映射（ORM）

- Everything is object
- Fields
  - basic
    - char, text
    - boolean, integer, float
    - date, time, datetime
    - binary
  - selection, function, related
  - relational
    - one2one
    - one2many
    - many2one
    - Many2many

```
from osv import fields
```

# ORM - Fields - Basic

- string
- boolean
- integer and float
- date, time and datetime
- binary (File)

## ORM - Fields - Basic - Char & Text

```
_columns = {  
    'name': fields.char('Name', size=32),  
    'description': fields.text('Description'),  
}
```

# ORM - Fields - Basic - Boolean & Integer & Float

```
_columns = {  
    'active': fields.boolean('Active'),  
    'level': fields.integer('Level'),  
    'price': fields.float('Price', digits=(16, 2)),  
}  
  
_defaults = {  
    'active': lambda *a: False,  
    'level': lambda *a: 1,  
    'price': lambda *a: 1.0,  
}
```

## 注意:

对象字段 `active == False` 的Record会自动被系统过滤掉，不会在画面上显示（相当于“软删除”）。

# ORM - Fields - Basic - Date & DateTime & Time

```
_columns = {  
    'birthdate': fields.date('BirthDate'),  
    'create_at': fields.datetime('Date & Hour'),  
    'time': fields.time('Hour'),  
}  
  
_defaults = {  
    'birthdate': lambda *a: time.strftime('%Y-%m-%d'),  
    'create_at': lambda *a: time.strftime('%Y-%m-%d %H:%M:%S'),  
    'time': lambda *a: time.strftime('%H:%M:%S'),  
}
```

# ORM - Fields - Basic - Binary

```
_columns = {  
    'image': fields.binary('Image')  
}
```

## ORM - Fields - Extended - Selection

```
_columns = {  
    'state': fields.selection(  
        [('draft', 'Draft'), ('confirmed', 'Confirmed')],  
        'State' ),  
}  
  
_defaults = {  
    'state': lambda *a: 'draft',  
}
```

注：定义state有两个状态：Draft 和 Confirm，在界面上的显示通常是一个下拉框。



## ORM - Fields - Extended – Function(1)

```
def _get_average_value( self, cr, uid, ids, name, args, context=None):  
    pass  
  
_columns = {  
    'avg': fields.function ( _get_average_value, fcnt_inv=_something_write,  
                             fcnt_search=_something_search, method=True, string= "Fields" ,  
type= "float" , store=True )  
}
```

注:

`fields.function(fnct, arg=None, fnct_inv=None, fnct_inv_arg=None, type="float",  
fnct_search=None, obj=None, method=False, store=False, multi=False)`

**fnct** : 预先定义的函数，返回本字段值。

**fnct\_inv** : 预定义的函数，保存本字段值到数据库。

**type** : 函数返回的本字段类型，可以是function以外的任何字段类型。

**fnct\_search** : 定义本字段的搜索行为。

**method** : **True**表示本字段函数来自于对象的方法，否则，来自于全局函数。

**store** : 是否保存本字段到数据库，默认值是 **False**。

**multi** : 组名，**multi** 组名相同的字段值会一次性计算。

## ORM - Fields - Extended – Function(2)

```
store = {  
    'object_name': ( function_name, ['field_name1', 'field_name2'], priority)  
}
```

注：

Store的含义是，当对象‘object\_name’的字段值[‘field\_name1’, ‘field\_name2’]发生变化时，系统自动调用函数function\_name， function\_name返回应重新计算的本对象（不是‘object\_name’对象）的记录ids，该ids会传给fnct\_inv函数重新计算字段值。

## ORM - Fields - Extended – Property

```
class res_partner(osv.osv):
    _name = "res.partner"
    _inherit = "res.partner"
    _columns = {
        'property_product_pricelist':
            fields.property( 'product.pricelist', type='many2one', relation='product.pricelist',
                             string="Sale Pricelist", method=True, view_load=True,
group_name="Pricelists Properties" ),
    }
```

```
<record model="ir.property" id="property_product_pricelist">
    <field name="name">property_product_pricelist</field>
    <field name="fields_id" search="[(('model','=', 'res.partner'),
    ('name','=', 'property_product_pricelist'))]" />
    <field name="value" eval="product.pricelist, '+str(list0)'" />
</record>
```

注：

本例相当于在对象res.partner上增加关联到对象product.pricelist的many2one的字段property\_product\_pricelist。但和many2one类型不同的是，本字段不是存储在对象res.partner上，而是在另外一张关系表ir.property中。

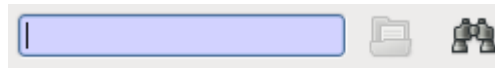
# ORM - Fields - Relational - one2many



```
class openacademy_instructor(osv.osv):  
    _name = 'openacademy.instructor'  
    _columns = {  
        'training_ids': fields.one2many(  
            'openacademy.training', 'instructor_id', 'Trainings' )  
    }
```

注：一门培训课程有多个培训师，课程和培训师间是一对多(one2many)的关系。

# ORM - Fields - Relational - many2one



```
_columns = {  
    'instructor_id': fields.many2one('openacademy.instructor',  
    'Instructor' )  
}
```

注：多个培训师提供一门课程。

# ORM - Fields - Relational - many2many

Name	City	Country	Type	Code	# of Contacts

```
_columns = {  
    'participant_ids' : fields.many2many('openacademy.participant',  
    'openacademy_training_participant_rel', 'training_id', 'participant_id',  
    'Participants' )  
}
```

- 一位学生可以参加多门培训课程
- 一门培训课程可以有几位学生

## ORM - Fields - Relational - related

```
_columns = {  
    'fiscalyear_id': fields.many2one('account.fiscalyear', 'Fiscal Year', required=True,  
    'company_id': fields.related('fiscalyear_id', 'company_id', type='many2one',  
relation='res.company', string='Company', store=True, readonly=True)  
}
```

注：

该related定义中，系统根据第一、第二个参数的'fiscalyear\_id'，'company\_id' 找到对象  
'account.fiscalyear'的字段'company\_id'所引用的'res.company'的记录，从而得到公司对象。

# ORM - Fields – Attributes(1)

- **Attributes 说明**

- **size**: 字段长度
- **digits**: 用于浮点型, 说明整数和小数部分位数
- **select**: 本字段是否用于过滤条件 (“True”|“False”, 1, 2 )
  - **select=1 or True**: 本字段用于一级过滤条件
  - **select=2**: 本字段用于二级过滤条件
- **required**: 必须
- **readonly**: 只读
- **domain**: **many2one**、**many2many**字段中, 用于选择关联对象的过滤条件。
- **string**: 字段显示名
- **translate**: 本字段可翻译
- **change\_default**: 别的字段的缺省值是否可依赖于本字段, 缺省值为: **False**。例子(参见 **res.partner.address**, **City**字段默认值依据**Zip**而变),
- **help**: **Tooltip**
- **states**: 定义特定**state**才生效的属性

更多参考: <http://shine-it.net/index.php/topic,2159.15.html>



## ORM - Fields – Attributes(2)

- **Attributes 说明**

- **context**: 在context 中增加一些变量，这些变量可用于on\_change方法及domain条件式。

- **on\_change**: 当本字段值改变时，调用Server端函数。例子：

- on\_change="onchange\_shop\_id(shop\_id)".

- **relation**: 当本字段是一个引用其他数据表的id 时，指定关联数据表名。通常用在related 和 function 类型的字段中。

- **级联关系:**

- **onDelete**

- **set null**: 删除主记录时候，从记录到主记录的引用置为null。

- **set default**: 删除主记录时候，从记录到主记录的引用置为缺省值。

- **cascade**: 删除主记录时候，级联删除从记录。

- **restrict**: 如果有从记录，不允许删除主记录。

- **no action**: 不采取任何动作，即删除主记录，但保留从记录不变。

更多参考: <http://shine-it.net/index.php/topic,2159.15.html>

domain 是OpenERP广泛使用的过滤/搜索条件。

domain 操作符

- '<>', '!=', '>=', '<=', '<', '>'
- 'like', 'ilike', 'not like', 'not ilike'
- 'in', 'child\_of'

domain 关系操作符

- & (*AND*)
- | (*OR*)
- ! (*NOT*)

# ORM - domain 简单示例

- 简单Domain示例:

```
domain = [('user_id', '=', user_id), ('state', '=', 'installed')]  
>>> 'user_id' = user_id AND 'state' = 'installed'
```

## 注意:

domain表达式中，左边是对象的字段名，该字段名不可以使用"."操作符，只可以是对象的直接字段。如不能是"user\_id.name"。右边是值，可以有"."，如user\_id.id。

# ORM - domain 复杂示例

- 复杂Domain示例:

```
[('reconcile_id', '=', False),  
 ('account_id.type', '=', 'payable'),  
 ('amount_to_pay', '>', 0),  
 '|',  
 ('date_maturity', '<', '2009-01-01'),  
 ('date_maturity', '=', False)]
```

等同于

`reconcile_id = 'f'`

`AND`

`account_id.type = 'payable'`

`AND amount_to_pay > 0 AND`

`( date_maturity < 2009-01-01 OR date_maturity = 'f' )`

注:

关系操作符按波兰表达式运算，默认操作符是 `AND` 。

# ORM - Special methods - Predefined Methods

osv
<pre>+create(cr,uid,vals:dict,context:{}=None): integer +read(cr,uid,ids:[],fields:[],context:{}=None): dict +write(cr,uid,vals:dict,context:dict=None) +unlink(cr,uid,ids:[],context:{}=None) +search(cr,uid,domain:[],offset,limit,order,context,count): [] +browse(cr,uid,domain:[],context:{}=None): array +copy(cr,uid,id,default=None,context:dict=None) +name_get(cr,uid,ids,context:dict=None): array +name_search(cr,uid,name,args,operator,context:dict=None,limit)</pre>

- 预定义的方法
- create:创建记录
- write:更新记录
- unlink:删除记录
- read:读取记录中的字段
- copy :复制记录
- search: 搜索记录
- browse: 通过搜索标准搜索记录
- name\_get:仅返回名称标识的记录
- name\_search:基于名称在相关的领域搜索
- init : **\_auto = False** 的情况，通常重载该方法创建数据库视图。
- \_auto\_init : 通常重载该方法创建数据库索引或SQL对象

# ORM - Special methods - create

- 该方法创建新记录（在数据表中Insert一条记录）

```
def create(self, cr, uid, vals, context=None):  
    pass
```

- Example 1

```
obj = self.pool.get('openacademy.instructor')  
id = obj.create(cr, uid, {'name': 'Stephane Wirtel', 'email': 'stephane@tinyerp.com'})
```

- Example 2 : 重载res.users 对象的Create方法，限制免费版最多创建20个用户。

```
def create ( self, cr, uid, vals, context=None):  
    USER_LIMITED = 20  
    cr.execute('select count(1) as number from ' % (self.pool.get('res.users').table, ))  
    value = cr.fetchone()[ 'number' ]  
    if value < USER_LIMITED:  
        return osv.osv.create(self, cr, uid, vals, context=context)  
    else:  
        raise osv.except_osv(("Warning"), ("Don't forget to call your integrator to pay the full  
contract !!!"))
```

# ORM - Special methods - write

- 更新记录，相当于执行数据库的Update命令

```
def write(self, cr, uid, ids, vals, context=None):  
    pass
```

- Example

```
obj = self.pool.get( 'openacademy.instructor' )
```

```
obj.write(cr, uid, ids, {'active':True})
```

# ORM - Special methods - unlink

- 删除记录，相当于执行数据库的Delete指令

```
def unlink(self, cr, uid, ids, context=None):  
    pass
```

- Example

```
obj = self.pool.get('openacademy.instructor')  
obj.unlink(cr, uid, ids)
```



## ORM - Special methods - read

- 读取指定记录的指定字段

```
def read(self, cr, uid, ids, fields, context=None):  
    pass
```

- Example : 读取id in ids 的记录的 name 和 email 字段

```
obj = self.pool.get('openacademy.instructor')  
records = obj.read(cr, uid, ids, ['name', 'email'])
```

```
for name, email in records:  
    pass
```

## ORM - Special methods - copy

- 复制指定id的记录，并插入数据库。如果只想复制字段值，不要在数据库中插入新记录，
- 使用方法 `copy_data`。Default 指定新记录的默认值

```
def copy(self, cr, uid, id, default = None, context = None):  
    pass
```

# ORM - Special methods - search

- 使用Domain条件搜索

```
def search(self, cr, uid, domain, offset=0,  
          limit=None, order=None, context=None, count=False):  
    pass
```

- Example

```
obj = self.pool.get('openacademy.instructor')  
ids = obj.search(cr, uid, [('email', '=', 'stephane@tinyerp.com')])
```

# ORM - Special methods - browse

- 浏览指定id或ids的对象

```
def browse( self, cr, uid, select, context=None, list_class=None, fields_process=None) :  
    pass
```

- Example

```
obj = self.pool.get( 'openacademy.instructor' )  
ids = obj.search(cr, uid, [('email','=', 'stephane@tinyerp.com')])  
instructors = obj.browse(cr, uid, ids)
```

```
for instructor in instructors:  
    print "name: %s" % instructor.name  
    print "email: %s" % instructor.email
```

## ORM - Special methods - name\_get

- 取得对象的name，默认情况是返回name字段，但可以重载本方法改写。

```
def name_get( self, cr, uid, ids, context=None):  
    pass
```

- Example: 不仅返回name字段，还返回id

```
obj = self.pool.get('openacademy.instructor')  
names = obj.name_get(cr, uid, ids )
```

```
for id, name in names:  
    pass
```

## ORM - Special methods - name\_search

- 根据名称或domain条件搜索记录，返回记录名称列表

```
def name_search( self, cr, uid, name= '', domain=None, operator= "ilike",  
context=None, limit = 100 ):  
    pass
```

- Example

```
obj = self.pool.get('openacademy.instructor')  
names = obj.name_search(cr, uid, domain=[('email', '=', 'stephane@tinyerp.com')])
```

## ORM - Special methods - init & \_auto\_init

- `init(cr)`
  - `_auto = False`
  - 创建数据库视图
- `_auto_init`
  - 创建数据库索引等对象

## ORM - Predefined Fields

默认情况下，OpenERP新建对象时候，会在数据表中自动添加下述字段。

- id: 对象id，是默认主键
- create\_uid: 记录创建者的用户id
- create\_date:记录创建时间
- write\_uid:记录更新者的用户id
- write\_date:记录更新时间



# ORM - Special Fields (1)

- name

- Char

```
_columns = {  
    'name' : fields.char('Name', size=64)  
}
```

- Sequence

- Integer

```
_columns = {  
    'sequence' : fields.integer('Sequence')  
}
```

```
_defaults = {  
    'sequence' : lambda *a: 0,  
}
```

## ORM - Special Fields (2)

- active

- Boolean

```
_columns = {  
    'active': fields.boolean('Activable')  
}
```

```
_defaults = {  
    'active': lambda *a: True,  
}
```

- state

- Selection

```
_columns = {  
    'state': fields.selection([ ('draft', 'Draft'), ('confirmed',  
    'Confirm') ], 'State')  
}
```

```
_defaults = {  
    'state': lambda *a: 'draft',  
}
```

# ORM - Object Declaration

OpenERP的对象属性有：

- 必须属性
  - **\_name**
  - **\_columns**
- 可选属性
  - `_table`
  - `_description`
  - `_defaults`
  - `_order`
  - `_rec_name`
  - `_auto`
  - `_constraints`
  - `_sql_constraints`
  - `_inherit`
  - `_inherits`

对象定义的详细参考：<http://shine-it.net/index.php/topic,2159.0.html>

## ORM – Object Declaration - \_name

指定对象名称，该名称用于从对象池中取得对象的Key值，其格式通常是“module\_name.class\_name”:

```
_name = 'openacademy.instructor'
```

提示:

对象对应的数据表名是，对象名中的“.”换成“\_”即得表名。

## ORM - Object Declaration - `_columns`

定义对象的属性，也是数据表中的字段。

```
_columns = {  
    'name' : fields.char( 'Name', size=64 ),  
}
```

## ORM - Object Declaration - \_table

指明对象对应的数据库表名。默认情况是对象名中"."换成"\_"即得表名，但也可以显式指定表名。

```
_table = 'openacademy_instructor'
```

## ORM - Object Declaration - \_description

对象描述，可以是任意文字

```
_description = 'Instructor'
```

## ORM - Object Declaration - \_defaults

定义字段的默认值，可以通过函数智能的取得默认值。

```
def _compute_date( self, cr, uid, ids, context = None ):  
    pass
```

```
_defaults = {  
    'active' : lambda *a: True,  
    'date' : _compute_date,  
}
```



## ORM - Object Declaration - \_order

指定过滤出来的记录的排序方法，下例按**date**字段降序排列

```
_order = 'date desc'
```

## ORM - Object Declaration - `_rec_name`

指定`name_get`方法返回哪个字段的值，默认情况是返回对象的 `name` 字段值，但可用本声明修改默认情况。

```
_rec_name = 'title'
```

## ORM - Object Declaration - `_auto`

模块安装时候，是否为对象自动创建数据表，默认是**True**，即自动创建。有时候，例如对象用于显示来自数据库视图的值时候，不需要创建数据表，应指定为**False**。

## ORM - Object Declaration - `_constraints`

指定记录插入、更新时候的检查条件。下例表示调用方法`check_date`检查字段`startdate`, `enddate` 的值, 如果`check`不通过, 报错“结束日期必须大于起始日期”。

```
def check_date( self, cr, uid, ids ):  
    pass  
_constraints = [  
    (check_date, '结束日期必须大于起始日期', ['startdate', 'enddate'] ),  
]
```

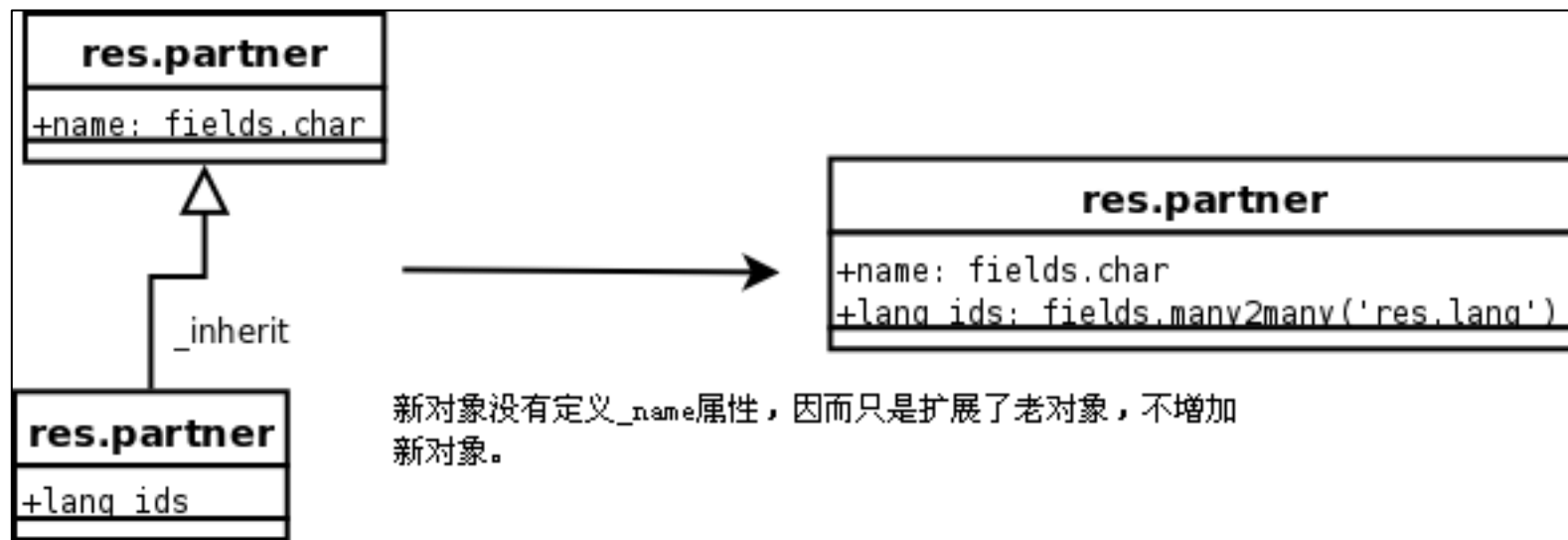
## ORM - Object Declaration - `_sql_constraints`

定义数据库约束条件。下例相当于在数据表上增加唯一性约束条件 `unique(name)`，如果违反约束条件，报错“名称必须唯一！”

```
_sql_constraints = [  
    ( 'uniq_name' ,    'unique(name)' ,    '名称必须唯一!' ),  
]
```

# ORM - Object Declaration - \_inherit (1)

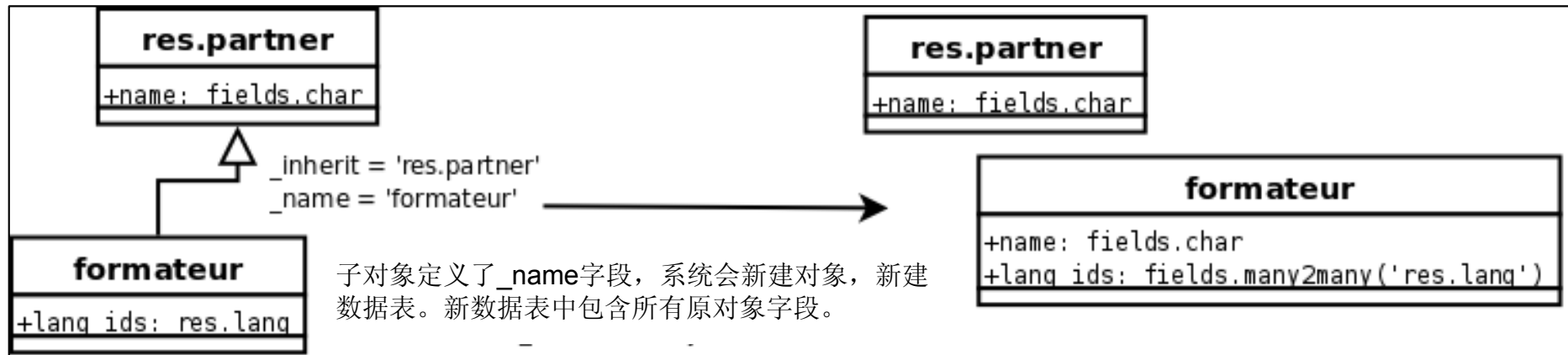
对象继承方式一：扩展现有对象。不新增Table，而是在原Table中增加字段。



```
class res_partner_add_langs(osv.osv):  
    _inherit = 'res.partner'  
    _columns = { 'lang_ids' : fields.many2many('res.lang', 'res_lang_partner_rel',  
        'partner_id', 'lang_id', 'Languages'),  
    }  
res_partner_add_langs()
```

## ORM – Object Declaration - \_inherit (2)

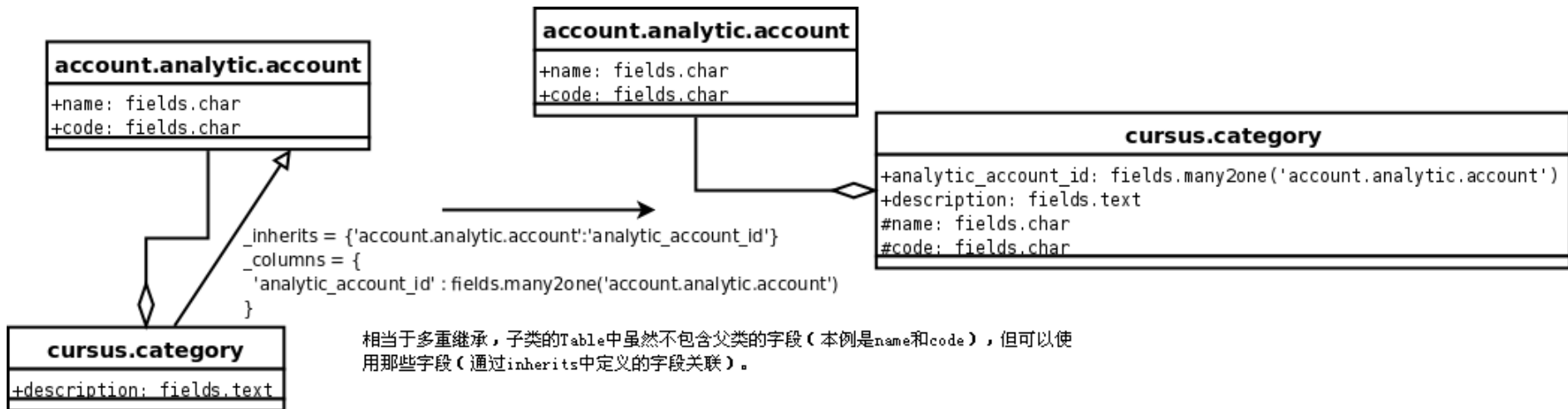
继承方式二：新增一个对象，该对象拥有原对象的所有方法和属性。



```
class formateur(osv.osv):  
    name = 'formateur'  
    _inherit = 'res.partner'  
    _columns = {  
        'lang_ids': fields.many2many('res.lang', 'res_lang_partner_rel', 'partner_id',  
        'lang_id', 'languages'),  
    }  
formateur()
```

## ORM – Object Declaration – \_inherits (3)

继承方式三：创建一个新对象，拥有原对象所有方法和属性。但新对象的数据表中不包括原对象的字段，仅仅包括新字段。



```
class cursus_category(osv.osv):  
    _name = 'cursus.category'  
    _inherits = { 'account.analytic.account' : 'analytic_account_id' }  
    _columns = { 'analytic_account_id' : fields.many2one(  
'account.analytic.account', 'Analytic Account' ),  
        'description' : fields.text('Description')  
    }  
cursus_category()
```



# ORM - View Mapping

创建数据库视图对象。

- `_auto = False`
- 声明对象字段
- 重载 `init`

```
1 def init(self,cr):
2     cr.execute("""
3         create or replace view VIEW_NAME as (
4             ...
5         )
6     """)
```

# XML Format

- 模块中有多种用途的XML文件，但所有XML文件的语法结构都是一样的

- module\_name\_data.xml
- module\_name\_view.xml
- module\_name\_wizard.xml
- module\_name\_workflow.xml
- module\_name\_report.xml
- module\_name\_demo.xml

# XML Format - Example

```
<?xml version="1.0"?>
<openerp>
  <data nouupdate="1">
    ...
  </data>
</openerp>
```

- *nouupdate="1"* 表示，模块升级时候不更新本文件数据。

# View – 概述（1）

## ■ 什么是视图（View）

Object用于存储业务数据，View用于向用户展现数据以及输入数据。View的构成包括field, separator, group, button等用户界面设计元素。

## ■ 视图的类型

- 列表(Tree View)
- 表单(Form View)
- 查询栏(Search View)
- 日历(Calendar View)
- 框图(Diagram View)
- 图形(Graph View)
- 甘特图(Gantt View)

## View – 概述（2）

★ Search: Manufacturing Orders ?

PENDING

READY

IN PRODUCTION

LATE

Reference :

Product :

Routing ? :

Source Document ? :

Group By...

Search

Clear

Search View

-- Filters --

Manufacturing Orders

New

1 - 2 of 2

	REFERENCE	SCHEDULED DATE	PRODUCT	PRODUCT QTY	PRODUCT UOM	ROUTING	TOTAL HOURS	TOTAL CYCLES	SOURCE DOCUMENT	STATE	
<input type="checkbox"/>	MO/00002	06/30/2011 07:55:06	[CPU_GEN] Regular processor config	1.00	PCE	Component Manufacturing	3.00	0.50	:MO/00001	Ready to Produce	×
<input type="checkbox"/>	MO/00001	07/02/2011 07:55:06	[PC1] Basic PC	1.00	PCE	Assembly Line 1	2.00	1.00		Waiting Goods	×
				2.00			5.00	1.50			

Tree View

## View – 概述（3）

### Manufacturing Orders ?

Reference : MO/00002

Reference :  Scheduled date :  Source Document ? :

Product :  Product Qty :  Product UOM :

**Consumed Products** **Finished Products** **Work Orders** **Scheduled Products** **Extra Information**

Bill of Material :  Routing ? :

Raw Materials Location ? :  Finished Products Location ? :

#### Products to Consume

1 - 4 of 4

PRODUCT	QTY	UOM	SOURCE LOC.	
[CPU1] Processor AMD Athlon XP 1800+	1.000	PCE	Stock	<input type="button" value="x"/>
[MB1] Mainboard ASUSTek A7N8X	1.000	PCE	Stock	<input type="button" value="x"/>
[FAN] Regular case fan	1.000	PCE	Stock	<input type="button" value="x"/>

#### Consumed Products

0 - 0 of 0

PRODUCT	QTY	UOM	PRODUCTION LOT

# View - File Description

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <openerp>
3    <data>
4      <!-- views -->
5      <record model="ir.ui.view" id="openacademy_training_form">
6        ...
7      </record>
8      <!-- actions -->
9      <record model="ir.actions.act_window" id="openacademy_training_act">
10        ...
11      </record>
12    <!-- menuitem -->
13    <menuitem name="Tools" id="openacademy_training_mi"
14              action="openacademy_training_act" />
15    <!-- 快捷键 V6.0新加的 -->
16    <shortcut name="Draft Purchase Order (Proposals)" model="purchase.order"
17             logins="demo" menu="m"/>
18  </data>
19 </openerp>
```

# View - View - Form

```
1  <record model="ir.ui.view" id="openacademy_training_form">
2    <field name="name">openacademy.training</field>
3    <field name="model">openacademy.training</field>
4    <field name="type">form</field>
5    <field name="arch" type="xml">
6      <form string="Training">
7        <field name="name" select="1" />
8        <field name="date_start" select="1" />
9      </form>
10    </field>
11  </record>
```



## View - View - List

```
1  <record model="ir.ui.view" id="openacademy_training_tree">
2    <field name="name">openacademy.training</field>
3    <field name="model">openacademy.training</field>
4    <field name="type">tree</field>
5    <field name="arch" type="xml">
6      <tree string="Trainings" >
7        <field name="name" />
8        <field name="date_start" />
9      </tree>
10   </field>
11 </record>
```

注：V6.x 中 增加了 **color**, **toolbar** 两个属性，例：

```
<tree colors="blue:usage=='view';darkred:usage=='internal' " >
  <field name="usage"/>
  ...
</tree>
```

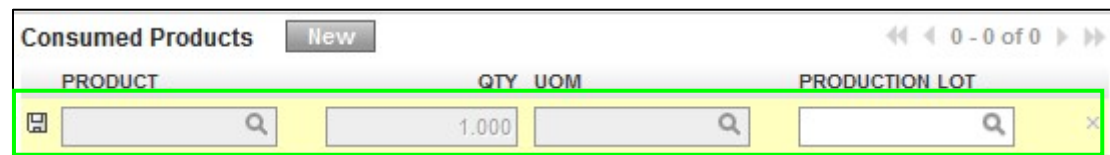
# View - View - Editable List

Editable实现记录的列表原地编辑

```
<tree string="Trainings" editable="bottom">
```

可选值:

- bottom: 新记录在列表的最后插入
- top: 新记录在列表的最上面插入



The screenshot shows a web interface for 'Consumed Products'. At the top, there is a title 'Consumed Products' and a 'New' button. To the right, there are navigation controls showing '0 - 0 of 0'. Below the title, there is a table with the following columns: 'PRODUCT', 'QTY', 'UOM', and 'PRODUCTION LOT'. The first row of the table is highlighted in yellow, indicating it is selected for editing. The 'PRODUCT' column contains a search icon and a text input field. The 'QTY' column contains the value '1.000'. The 'UOM' column contains a search icon and a text input field. The 'PRODUCTION LOT' column contains a search icon and a text input field. A green border highlights the entire row, and a small 'x' icon is visible in the bottom right corner of the row.

# View - View - Tree (1)

- 层次结构的树形视图 (parent\_id, child\_ids)

```
<record model="ir.ui.view" id="openacademy_category_tree">
  <field name="name">openacademy.category.tree</field>
  <field name="model">openacademy.category</field>
  <field name="field_parent">child_ids</field>
  <field name="type">tree</field>
  <field name="arch" type="xml">
    <tree string="Categories">
      <field name="name" />
    </tree>
  </field>
</record>
```

LOCATION NAME
Shop 1
Shop 2
▼ 上海联城超市
Output
▼ Stock
A库
B库
Shelf 1
Shelf 2

## View - View - Tree (2)

- 用于Action中的列表视图

```
<record model="ir.actions.act_window" id="openacademy_category_tree_act">  
<field name="name">All Categories</field>  
<field name="res_model">openacademy.category</field>  
<field name="view_type">tree</field>  
<field name="view_mode">tree, form</field>  
<field name="domain">[( 'parent_id', '=', False)]</field>  
</record>
```

- 此例的Domain 过滤出所有一级目录(没有父亲)。

## View - View - Tree (3)

下例实现“展开按钮”显示子目录。

```
<record model="ir.actions.act_window" id="cursus_by_category_act">
<field name="res_model">openacademy.cursus</field>
<field name="view_type">form</field>
<field name="view_mode">tree, form</field>
<field name="domain">[( 'category_id', 'child_of', [active_id])]</field>
</record>
```

```
<record model="ir.values" id="ir_action_cursus_by_category">
<field name="key2" eval="'' tree_but_open' " />
<field name="model" eval="'' openacademy.category' " />
<field name="name">Cursus</field>
<field name="value" eval="'' ir.actions.act_window,%d' %cursus_by_category_act" />
<field name="object" eval="True" />
</record>
```

## View - View - Graph

```
<record model="ir.ui.view" id="view_id">
  <field name="name">view_name</field>
  <field name="model">object_name</field>
  <field name="type">graph</field>
  <field name="arch" type="xml">
    <graph string="Graph Title" type="pie">
      <field name="first_field"/>
      <field name="second_field" operator="+" />
    </graph>
  </field>
</record>
```

- graph 的 type 属性可选值:
  - pie 饼图
  - bar 柱状图

# View - View - Calendar

```
<record model="ir.ui.view" id="openacademy_training_calendar">
  <field name="name">openacademy.training.calendar</field>
  <field name="model">openacademy.training</field>
  <field name="type">calendar</field>
  <field name="arch" type="xml">
    <calendar string="Training" date_start="date_start" date_stop="date_stop">
      <field name="name" />
    </calendar>
  </field>
</record>
```

- date\_start 开始日期
- date\_stop 结束日期 (可选)

# View - Action

1  
2  
3  
4  
5  
6

```
<record model="ir.actions.act_window" id="openacademy_training_act">  
  <field name="name">Training</field>  
  <field name="res_model">openacademy.training</field>  
  <field name="view_type">form</field>  
  <field name="view_mode">tree,form</field>  
</record>
```



## View - MenuItem

```
1 <menuItem name="Tools"  
2           id="tools_menu" />  
3 <menuItem name="OpenAcademy"  
4           id="openacademy_training_menu"  
5           parent="openacademy.tools_menu"  
6           action="openacademy_training_act" />
```

# View - Widgets

- `<field />`
  - ComboBox
  - CheckBox
  - Date
  - Text
  - Integer, Float
- `<label />`
- `<separator />`
- `<button />`
- `<notebook />`

## View - Widgets (2)

```
<form>
    ...
    <field name="field_name" />
    ...
</form>
```

- 系统会根据要显示的字段类型，自动选择默认的Widgets

## View - Widget - Label

```
<form>  
...  
<label string= “我的标签” />  
...  
</form>
```

## View - Widget - Separator

```
<form>  
...  
<separator string= “标题 Separator”/>  
...  
</form>
```

## View - Widget – Button(1)

```
# object_view.xml
<form>
...
<button type="object" name="python_function" string="Label" />
<button type="workflow" name="workflow_signal" string="Label"/>
<button type="action" name="%(wizard_id)d" string="Label"/>
...
</form>

# object_wizard.xml
<wizard id="wizard_id"
keyword="client_action_multi"
model="openacademy.training"
name="openacademy.training.do_something"
multi="True"
string="Label"/>
```

## View - Widget – Button(2)

- **type** – 可选值有 **workflow** (default), **object**, **action**  
**workflow**表示点击按钮，发送**name**属性指定的**signal**。**object** 表示点击按钮，调用**name**属性指定的对象方法。**action** 表示点击按钮，调用**name**属性指定的动作(**ir.actions.actions**)
- **special** – 目前只有一个值：**cancel**，表示本按钮关闭画面，不做任何进一步动作。注意：**special** 和**name** 是互斥的，不能同时出现。
- **name** – 指定哪个**signal**被发送，哪个方法被调用，哪个**action**被触发。
- **confirm** – 点击按钮时弹出一个确认**Message**
- **string** – 按钮的显示名
- **icon** – 按钮的显示图标
- **states**, **attrs**, **invisible**, **default\_focus** – 与**fields**的属性含义一致。

# View - Widget - Notebook

```
<form>
...
<notebook>
  <page string="First page">
    ...
  </page>
  <page string="Second page">
    ...
  </page>
</notebook>
...
</form>
```



# View - Widget - Group

```
<form>  
  ...  
  <group>  
    <field name="state" />  
    <button />  
    <button />  
  </group>  
</form>
```

# View – Attributes

- **select** : 可选值
  - 1 -> 一级过滤项目
  - 2 -> 二级（扩展）过滤项目
- **string** – 字段显示名
- **password** – 本字段内容是否以\*号显示
- **mode="tree,graph"**: one2many等复杂字段的编辑模式
- **nolabel** – 不显示标签
- **colspan** – 列宽
- **col** – 本字段的列宽
- **default\_focus** – 本字段是否获得默认焦点
- **states** – 指定什么状态下本字段可编辑, 如 **states="draft,confirm"**
- **domain** – 用于 (many2one, many2many) 关系中过滤关系对象
- **eval** – 通过一个Python表达式计算本字段的值
- **required** – 本字段是否必须的
- **readonly** – 本字段是否只读
- **Invisible** – 本字段是否可见
- **widget** – 本字段的界面元素

## Widget的可选界面元素:

one2one\_list  
one2many\_list  
many2one\_list  
many2many  
url  
email  
image  
float\_time  
reference

# Relate Window

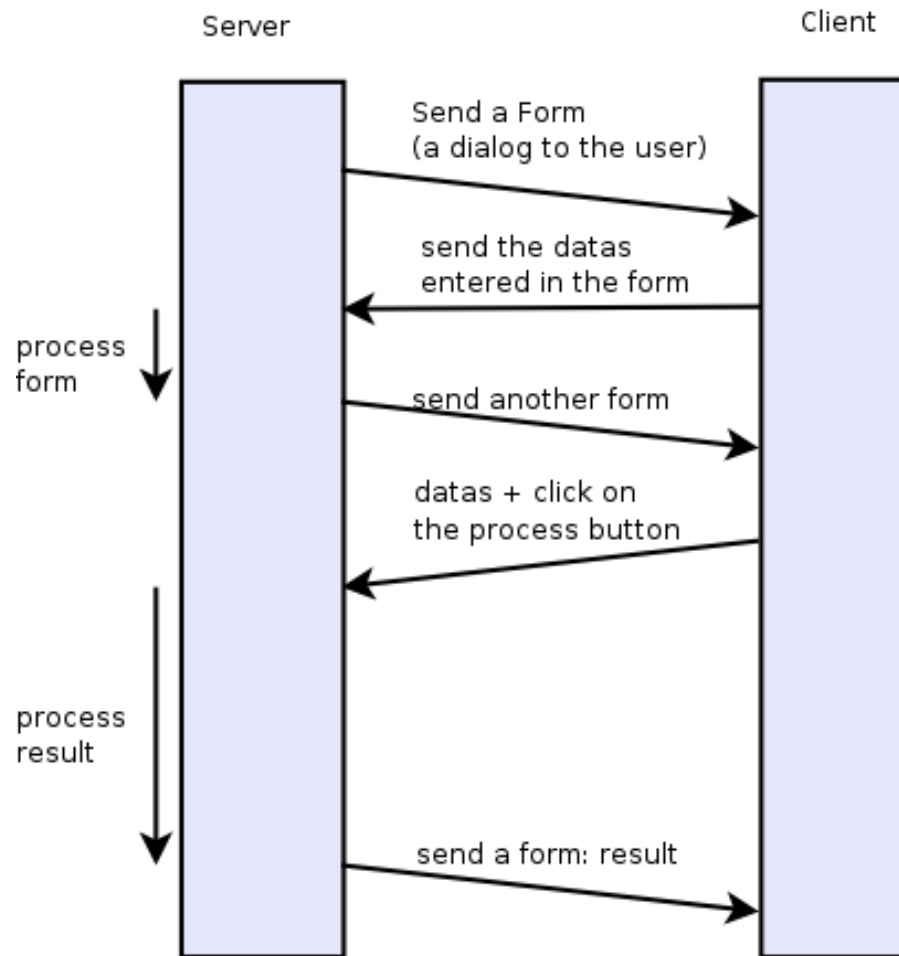
- Example:

```
<act_window id="openacademy_instructor_relate_its_training"  
  name="Trainings"  
  src_model="openacademy.instructor"  
  res_model="openacademy.training"  
  domain="[(instructor_id, '=', active_id)]"  
>
```

该实例弹出窗口，显示当前培训师所有的课程。

# Wizard

## A wizard process: example



### Wizard用于系统和用户的交互:

1. 系统发送一个表单给客户端。
2. 用户填写表单，客户端发送数据回系统。
3. 系统基于用户数据执行某些动作，而后结束，或者继续发送表单和用户交互。

## 老式向导(Wizard) – 模块结构

**Wizard**文件结构:

```
module_name/  
  module_name_wizard.xml  
  wizard/  
    __init__.py  
    wizard_name.py
```

```
# module_name/__init__.py  
import wizard
```

```
# module_name/wizard/__init__.py  
import wizard_spam.py
```

```
<!-- module_name_wizard.xml -->  
<?xml version="1.0" encoding="UTF-8" ?>  
<openerp>  
  <data>  
    <wizard id="openacademy_wizard_id" keyword="client_action_multi"  
model= "openacademy.training" name= "openacademy.training.openacademy_wizard"  
multi= "True" string= "Do Something" />  
  </data>  
</openerp>
```

注:

在对象openacademy.training表单右边工具条增加wizard

“openacademy.training.openacademy\_wizard”，显示菜单名为” Do Something”。

## 老式向导(Wizard) – 交互数据和视图

*# 定义与客户端交互需要的数据字段*

```
fields = { 'instructor_id' : { 'string' : 'Instructor' , 'type' :  
    'many2one' , 'relation' : 'openacademy.instructor' , 'required' : True },  
    'subject' : { 'string' : 'Subject' , 'type' : 'char' , 'size' : 64,  
    'required' : True },  
    'text' : { 'string' : 'Message' , 'type' : 'text' , 'required' : True },  
    'for_instructor' : { 'string' : 'Send to Instructor' , 'type' : 'boolean' ,  
    'default' : lambda *a: False }  
}
```

```
form = """<?xml version="1.0"?>  
<form string="Mass Mailing">  
<field name="instructor_id" /> <newline />  
<field name="subject" /> <newline />  
<field name="text" /> <newline />  
<field name="for_instructor" /> <newline />  
</form>  
"""
```

## 老式向导(Wizard) – 对象定义 - Wizard Class

```
class openacademy_wizard(wizard.interface):
    def _init(self, cr, uid, vals, context):
        return {'subject': "Default Subject"}
    def mass_mail_send(self, cr, uid, vals, context):
        return {}
    states = { 'init' : { 'actions' : [_init], 'result' : { 'type' : 'form', 'arch'
: form, 'fields' : fields, 'state' : [('end', 'Cancel'), ('send', 'Send')], }
        },
        'send' : { 'actions' : [mass_mail_send], 'result' : { 'type' :
'send', 'state' : 'end' }
        }
    }

openacademy_wizard('openacademy.training.openacademy_wizard')
```

注:

Wizard Class 定义交互的Step，上述例子有两个Step: init 和 send，第一个Step，系统发送定义好的Form 和数据字段给客户端。state在Form的右下方增加按钮: Cance 和 Send。用户点击Send，则进入下个Step: send。Step send 的Type为'send'，且其state为end，这表示交互结束。type的取值有两个: form 和 state。前者表示本Step要向客户端发送Form，后者表示，不发送Form，只执行服务端动作。

# 新式向导(Wizard)

数据定义：等同于普通对象的定义，唯一区别是，从osv.osv\_memory派生。  
class account\_chart(osv.osv\_memory):

表单定义：等同于普通Form的定义。

Step的定义：Form上的type='action'的button调出下一个act\_window。

和老式Wizard相比的优点有：

- 1.inheritance
- 2.workflows
- 3.complex relation fields
- 4.computed fields
- 5.all kind of views (lists, graphs, ...)



## 新式向导(Wizard) – 示例 (1)

```
class product_margin(osv.osv_memory):
    _name = 'product.margin'
    _description = 'Product Margin'

    def _action_open_window(self, cr, uid, ids, context):
        pass
    _columns = {
        'from_date': fields.date('From'),
        'to_date': fields.date('To'),
        ...
    }
    _defaults = {
        'from_date': lambda *a: time.strftime('%Y-01-01'),
        'to_date': lambda *a: time.strftime('%Y-01-01'),
    }
product_margin()
```

```
<act_window name="Open Margin"
res_model="product.margin"
src_model="product.product" view_mode="form"
target="new" key2="client_action_multi"
id="product_margin_act_window"/>
```

```
<record id="product_margin_form_view"
model="ir.ui.view" >
    <field name="name">product.margin.form
</field>
    <field name="model">product.margin </field>
    <field name="type">form </field>
    <field name="arch" type="xml">
        <form string="Properties categories">
            <separator colspan="4" string="General
Information" />
            <field name="from_date" />
            <field name="to_date" />
            <group col="4" colspan="2">
                <button special="cancel"
string="Cancel" type="object"/>
                <button
name="_action_open_window" string="Open
Margins" type="object" default_focus="1" />
            </group>
        </form>
    </field>
</record>
```

## 新式向导(Wizard) – 示例 (2)

定义Wizard的Action:

```
<act_window name="Open Margin" res_model="product.margin"
  src_model="product.product" view_mode="form"
  target="new" key2="client_action_multi"
  id="product_margin_act_window"/>
```

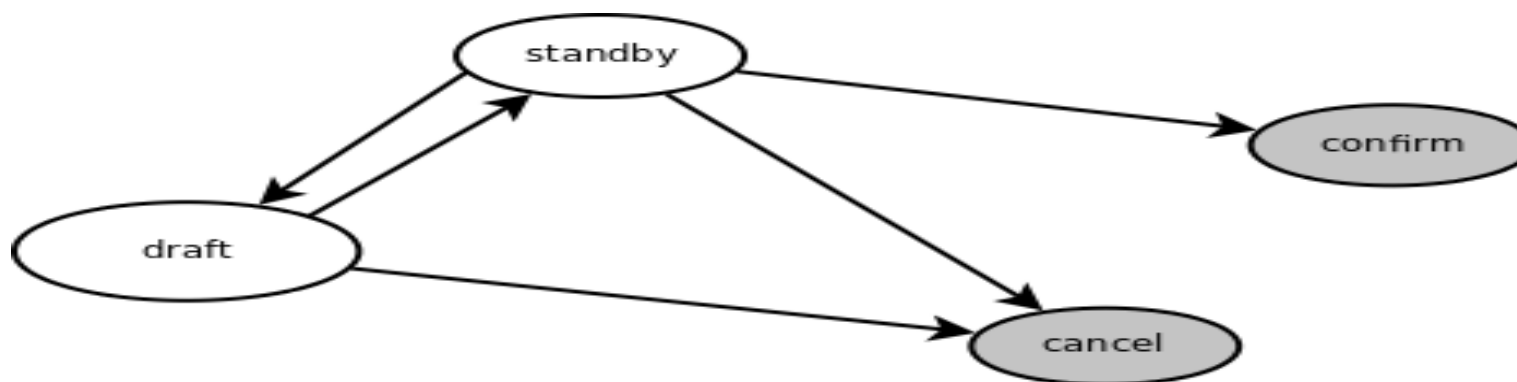
通过菜单弹出**Wizard**:

```
<menuitem id="main" name="OSV Memory Wizard Test"/>
<menuitem action="product_margin_act_window"
  id="menu_product_act"
  parent="main" />
```

通过按钮弹出**Wizard**:

```
<button name="% (product_margin.product_margin_act_window)d"
  string="Test Wizard" type="action" states="draft"/>
```

# Workflow



# Workflow - What is it ?

- 基于对象
- 声明式工作流
- 活动(Activity)
  - 开始节点
  - 结束节点
  - 中间节点
- 迁移(Transition)

关于工作流的详细信息，参看：<http://shine-it.net/index.php/topic,2494.0.html>

# Workflow - Declaration

```
<record model="workflow" id="wkf_openacademy">  
<field name="name">openacademy.wkf</field>  
<field name="osv">openacademy.training</field>  
<field name="on_create">True</field>  
</record>
```

注:

- 1) **osv** 指定本工作流操作的对象，是一个OpenERP的对象Name
- 2) **on\_create**: 系统新建一个对象实例时候，是否同时创建该对象实例的工作流实例。

# Workflow - Activities

- 开始节点

```
<record model="workflow.activity" id="act_draft">  
  <field name="wkf_id" ref="wkf_openacademy"/>  
  <field name="flow_start">True</field>  
  <field name="name">draft</field>  
  <field name="kind">function</field>  
  <field name="action">openacademy_training_draft()</field>  
</record>
```

- action种类:
  - Dummy
  - Stop All
  - subflow
    - 指定一个子工作流 (example: account.invoice)
  - function
    - 方法调用, 参看openacademy.training
- 一个工作流有且只允许有一个开始节点

# Workflow - Activities

- 结束节点

```
<record model="workflow.activity" id="act_close">  
<field name="wkf_id" ref="wkf_openacademy"/>  
<field name="flow_stop">True</field>  
<field name="name">close</field> <field name="kind">function</field>  
<field name="action">openacademy_training_close()</field>  
</record>
```

- 中间节点

```
<record model="workflow.activity" id="act_started">  
<field name="wkf_id" ref="wkf_openacademy" />  
<field name="name">started</field>  
<field name="kind">function</field>  
<field name="action">openacademy_training_started()</field>  
</record>
```

```
<record model="workflow.activity" id="act_standby">  
<field name="wkf_id" ref="wkf_openacademy" />  
<field name="name">standby</field>  
<field name="kind">function</field>  
<field name="action">openacademy_training_standby()</field>  
</record>
```

# Workflow - Transitions

```
<record model="workflow.transition" id="transition_1">  
  <field name="act_from" ref="act_draft" />  
  <field name="act_to" ref="act_close" />  
  <field name="signal">openacademy_to_close</field>  
</record>
```



# Workflow - Transitions - Methods (1)

```
class openacademy_training(osv.osv):
    _name = 'openacademy.training'
    _openacademy_training_states = [ ('draft', 'Draft'), ('started', 'Started'),
    ('standby', 'Standby'), ('closed', 'Closed'), ]
    _columns = {
        ...
        'state': fields.selection(_openacademy_training_states, 'State', readonly=True),
    }
    _defaults = {
        'state': lambda *a: 'draft',
    }

def openacademy_training_started(self, cr, uid, ids):
    print self.write(cr, uid, ids, {'state' : 'started'})

def openacademy_training_draft(self, cr, uid, ids):
    print self.write(cr, uid, ids, {'state' : 'draft'})

def openacademy_training_close(self, cr, uid, ids):
    print self.write(cr, uid, ids, {'state' : 'close'})

def openacademy_training_standby(self, cr, uid, ids):
    print self.write(cr, uid, ids, {'state' : 'standby'})
```

## Workflow - Transitions - Methods (2)

系统新建培训对象(**openacademy\_training**)时候是draft状态，培训开始的话，点击**start**按钮，系统调用方法openacademy\_training\_started，状态变为**started**。

1. openacademy\_training\_draft
2. openacademy\_training\_started

## Workflow - Transitions - Group

```
<record model="workflow.transition" id="transition_1">  
  <field name="act_from" ref="act_draft" />  
  <field name="act_to" ref="act_close" />  
  <field name="group_id" ref="openacademy_instructor"/>  
  <field name="signal">openacademy_to_close</field>  
</record>
```

# Workflow - Transitions - Conditions

```
<record model="workflow.transition" id="transition_1">
  <field name="act_from" ref="act_draft" />
  <field name="act_to" ref="act_close" />
  <field name="condition">...</field>
  <field name="signal">openacademy_to_close</field>
</record>
```

- 条件方法

```
<field name="condition">test_draft_to_started()</field>
```

*# openacademy.py*

```
def test_draft_to_started(self, cr, uid, ids):
    objs = self.browse(cr, uid, ids)
    for obj in objs:
        if obj.note1.find('do not start') > -1:
            return False
    return True
```

- Expression

```
<field name="condition">(order_policy=='prepaid') or ((order_policy=='postpaid') and
shipped)</field>
```

- SXW -> RML
- RML -> PDF
  - RML XML
  - ReportLab 渲染RML成 PDF
- OpenOffice Report Designer

```
tiny_sxw2rml.py training.sxw > training.rml
```

## Report (2)

```
module_name/  
    module_name_report.xml  
report/  
    __init__.py  
    report_name.py
```

## Report (3)

```
<report    id="report_openacademy_training"  
           string="Summary Of OpenAcademy"  
           model="openacademy.training"  
           name="openacademy.training.report"  
           auto="True"  
           rml="openacademy/report/openacademy_training.rml"  
           keyword="client_print_multi"  
           multi="True"  
           menu="True" />
```

## Report (4)

```
from report import report_sxw
import time
class openacademy_report(report_sxw.rml_parse):
    def __init__(self, cr, uid, name, context):
        super (openacademy_report, self).__init__(cr, uid, name, context)
        self.localcontext.update({
            'time' : time, 'instructor' : self._print_instructor,
        })

    def _print_instructor(self, instructor_id):
        return 'Toto'

openacademy_report('openacademy.training.report', 'openacademy.training',
'addons/openacademy/report/openacademy_training.rml', parser=openacademy_report,
header=True)
```



## Report (5)

```
<?xml version="1.0"?>
<document filename="test.pdf">
  <template pageSize="(595.0,842.0)" title="Test" author="Martin Simon" allowSplitting="20">
    <pageTemplate id="first">
      <frame id="first" x1="5.0" y1="57.0" width="590" height="828"/>
    </pageTemplate>
  </template>
  <stylesheet>
    <paraStyle name="Heading" fontName="Helvetica" fontSize="14.0" leading="17"
spaceBefore="12.0" spaceAfter="6.0"/>
    <paraStyle name="P1" fontName="Times-Roman" fontSize="6.0" leading="8" alignment="LEFT"/>
  </stylesheet>
  <story>
    <para style="Heading">
      <font face="Times-Roman" size="16.0"> Detail Training</font>
    </para>

    <para style="P1">
      <font face="Times-Roman" size="12.0">Module OpenAcademy</font>
    </para>
  </story>
</document>
```

## Report (6)

直接插入Python 代码的格式: `[[ et ]]`

```
<story>
  <para style="Heading">
    <font face="Times-Roman" size="16.0">Date: [[ time.strftime('%Y-%m-%d') ]]</font>
  </para>
```

class 'report' 中可以将报表需要的数据对象塞入**localcontext** 供RML使用。

```
class openacademy_report(report_sxw.rml_parse):
    def __init__(self, cr, uid, name, context):
        super(openacademy_report, self).__init__(cr, uid, name, context)
        self.localcontext.update({
            'time': time,
            'instructor': self._print_instructor,
        })
```

## Report - Available Objects

报表上下文 'localcontext' 中，系统默认包含了下述一些对象可供报表使用：

- user: 当前登录用户对象
- company: 当前用户所在公司对象
- repeatIn: Python方法 'repeatIn'
- setLang: 'setLang'方法
- setTag: 'setTag'方法
- removeParentNode: 'removeParentNode' 方法
- format: 'format' 方法
- formatLang: 'formatLang'方法
- logo: 公司logo
- lang: 公司lang

# Report - repeatIn

repeatIn 方法:

```
<story>
  <para style="Heading">
    <font face="Times-Roman" size="16.0"> Detail Training </font>
  </para>
  <para style="P2"> [[ repeatIn(objects,'o') ]]</para>
```

```
<story>
  <para style="Heading">
    <font face="Times-Roman" size="16.0"> Detail Training </font>
  </para>

  <section>
    <para style="P2"> [[ repeatIn(objects,'o') ]]</para>
  </section>
```

如何与其他系统沟通？

# Web-Services - Python

Import 包 xmlrpclib

```
import xmlrpclib
```

**注意：**

每次远程调用，都必须提供用户ID，密码和数据库名

# Web-Services - Python - Login

```
import xmlrpclib
```

```
database = "terp"
```

```
username = "admin"
```

```
password = "admin"
```

```
socket = xmlrpclib.ServerProxy( "http://localhost:8069/xmlrpc/common" )
```

```
user_id = socket.login( database, username, password )
```

# Web-Services - Python – search

```
socket = XMLRPC::Client.new( 'http://localhost', '/xmlrpc/object', 8069 )
ids = socket.execute( database, user_id, password, 'openacademy.instructor',
    'search', [] )
instructors = socket.execute( database, user_id, password, 'openacademy.instructor',
    'read', [ 'name', 'email' ] )

for instructor in instructors:
    print "instructor: %20s - %s" % ( instructor[ 'name' ], instructor[ 'email' ] )
```



# Web-Services - Ruby

Import 包 xmlrpc/client

```
require 'xmlrpc/client'
```

**注意：**

每次远程调用，都必须提供用户ID，密码和数据库名

# Web-Services - Ruby – Login

```
require 'xmlrpc/client'
```

```
database = "terp"
```

```
username = "admin"
```

```
password = "admin"
```

```
socket = XMLRPC::Client.new( 'http://localhost', '/xmlrpc/common', 8069 )
```

```
user_id = socket.login( database, username, password )
```

# Web-Services - Ruby - Search

```
socket = XMLRPC::Client.new('http://localhost', '/xmlrpc/object', 8069 )
ids = socket.execute( database, user_id, password, 'openacademy.instructor',
' search', [] )
instructors = socket.execute( database, user_id, password, 'openacademy.instructor',
' read', [' name', ' email' ] )

for instructor in instructors:
    print "instructor: %20s - %s" % [ instructor[' name'], instructor[' email' ] ]
```

## OpenERP调试 – Debug Level

```
LOG_NOTSET = 'notset'  
LOG_DEBUG_RPC = 'debug_rpc'  
LOG_DEBUG2 = 'debug2'  
LOG_DEBUG = 'debug'  
LOG_INFO = 'info'  
LOG_WARNING = 'warn'  
LOG_ERROR = 'error'  
LOG_CRITICAL = 'critical'
```

# OpenERP调试 – XMLRPC调试程序

下例调试对象‘**account.payment.term**’的方法‘**compute**’的输出值

```
# -*- encoding: utf-8 -*-
```

```
import xmlrpclib #导入xmlrpc库，这个库是python的标准库。
```

```
username = 'admin' #用户登录名
```

```
pwd = '123' #用户的登录密码，测试时请换成自己的密码
```

```
dbname = 'ch01' #数据库帐套名，测试时请换成自己的帐套名
```

```
# 第一步，取得uid
```

```
sock_common = xmlrpclib.ServerProxy('http://localhost:8069/xmlrpc/common')
```

```
uid = sock_common.login(dbname, username, pwd)
```

```
#replace localhost with the address of the server
```

```
sock = xmlrpclib.ServerProxy('http://localhost:8069/xmlrpc/object')
```

```
res = sock.execute(dbname, uid, pwd, 'account.payment.term', 'compute', 1, 2000)
```

```
print res
```

# Backup/Restore

- 备份数据库

```
pg_dump -b dbname -Fc -f filename.dmp
```

- 备份数据库文件

```
tar cf filestore.tar filestore/dbname
```

- 恢复数据库

```
pg_restore -d dbname filename.dmp
```

如何参与OpenERP社团？

## More Information !

OpenERP: <http://www.openerp.com>

OpenERP中文论坛: <http://shine-it.net/index.php>

中国OpenERP顾问社区: <http://openerp-china.org/>

OpenERP中文入门教程:  
<http://shine-it.net/index.php/topic,882.0.html>

Forum: <http://www.openerp.com/forum>

Wiki: <http://www.openerp.com/wiki>

LaunchPad: <http://www.launchpad.net/~openerp/>



# Contribution

As OpenERP is OpenSource, please feel free to contribute !!!

How to Contribute:

[http://openerp.com/wiki/index.php/Community:How To Contribute](http://openerp.com/wiki/index.php/Community:How_To_Contribute)

LaunchPad: <http://www.launchpad.net/~openerp/>