昆山一百计算机有限公司

Odoo 8.0 开发手册

第五版

作者：Amos

发布日期：2014-05-20

公司网站：http://www.100china.cn

Odoo 社区：http://www.openerpbbs.com

演示：http://www.Odoo.pw

购买 APP 地址：http://openerp.taobao.com/

QQ 技术交流群：249757342　　手册群：249941048（限购买用户）

目录

导读

本书介绍 Odoo 的架构与应用程序技术细节、构成 Odoo 的层、应用程序组件之间的通信方式和协议。总结开发语言和技术堆栈的一些细节。

Odoo 是一个支持多用户的三层架构 :数据库层进行数据存储 ,应用层进行处理和提供业务功能 ,表现层提供用户界面。在 Odoo 中，这些层是隔离的。应用程序层本身作为内核。可以安装多个附加模块，以便创建 Odoo 的特定实例，来适应具体需求。此外，Odoo 遵循模型 - 视图 - 控制器（MVC）架构模式

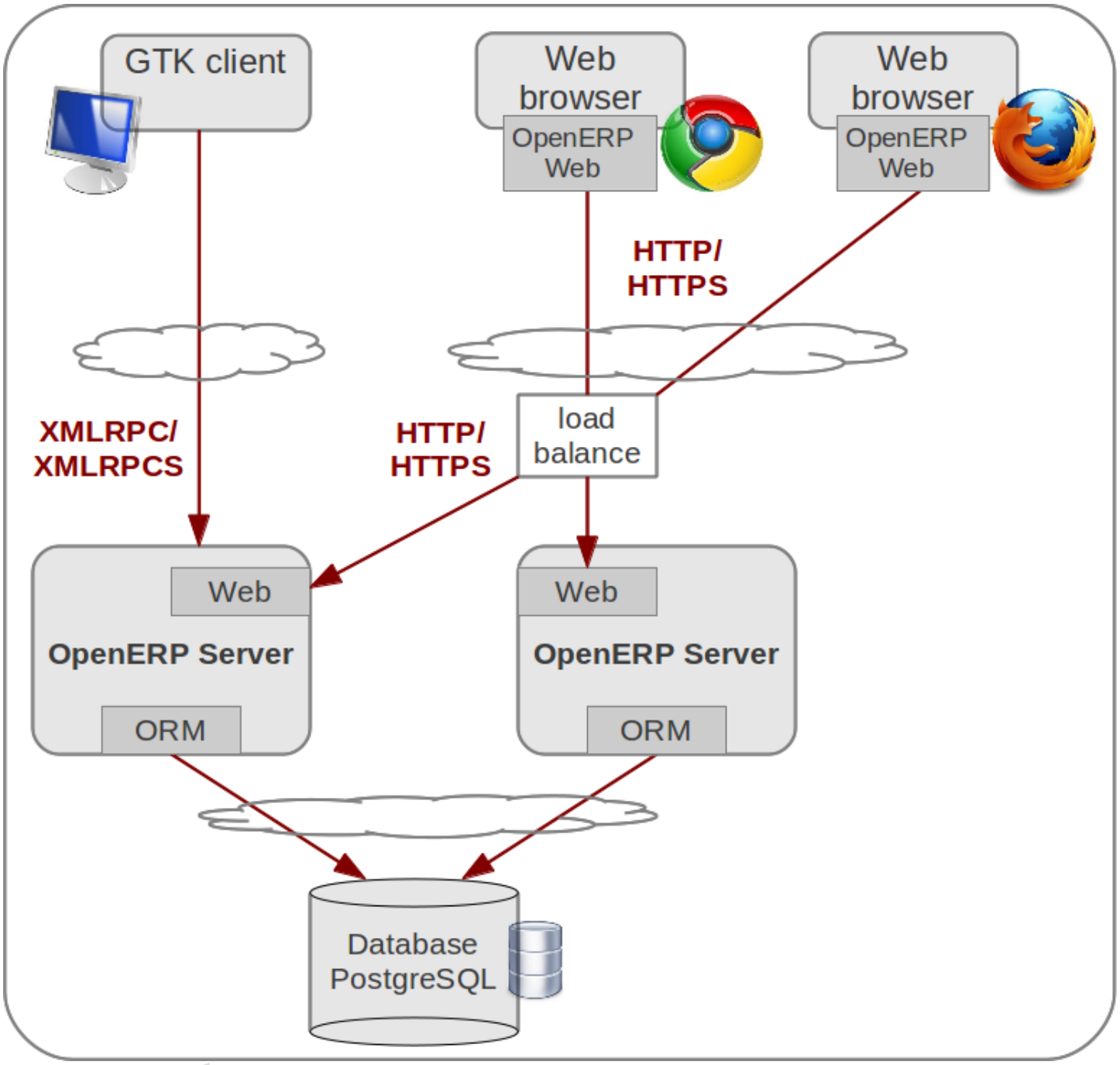
Odoo 系统由三个主要部分组成：

(一) PostgreSQL 数据库服务器包含所有 Odoo 数据库。数据库包含所有应用程序数据，以及 Odoo 系统主要的配置元素。请注意，这个服务器可以按集群数据库方式部署。

(二) Odoo 服务器包含所有的企业逻辑，确保 Odoo 的优化运行。其中一个层是 ORM 引擎，专门与 PostgreSQL 数据库的通信。另外一个层是 Web 层，控制服务器和 Web 浏览器通信。可部署多个服务，比如，结合负载平衡机制。

(三) 客户端运行在 web 浏览器中，是 JavaScript 应用程序。

数据库和 Odoo 的服务器可以安装在同一台计算机上，或者分布在单独的计算机服务器上，例如，出于性能方面的考虑。



Odoo 架构每个层的详细信息：

(一) Odoo 的数据层是关系型数据库 PostgreSQL。虽然从 Odoo 模块可直接执行 SQL 查询，但大多数都是通过 ORM 层访问关系数据库。

(二) 数据库包含所有应用程序数据，和大部分的 Odoo 系统配置要素。请注意，这个服务器可以使用集群数据库方式部署。

(三) Odoo 提供一个可以建立特定业务应用的应用程序服务，同时是一个完整开发框架，提供了一系列功能帮助编写那些应用程序。在这些功能中，Odoo 的 ORM 在 PostgreSQL 上提供的功能和接口。 Odoo 服务还具有一个特定的层，用来与基于 Web 浏览器的客户端通信。这一层用来连接服务器和使用标准浏览器的用户。

使用本教程开发一个真实的会议安排模块，并对各个功能点进行详细的讲解，内容包括：Odoo 对象、接口、视图、报表、工作流、安全性、向导、XML-RPC、翻译、性能优化等，快速开发一个应用，并对开发技巧进行深度解析。


帐套: demo
OpenERP
售前咨询：135 8493 5775
用户名：
密码：
登录
淘宝上购买APP
OpenERPBBS
OpenERP千人QQ群


消息
销售
POS
借入借出
项目
会员管理
会计
采购
仓库
生产
市场营销
人力资源
午餐管理
车辆管理
活动
知识管理
工具
报表
设置
Administrator
OpenERP
报价单
新建
或 导入
(1-1) 总 1
报价单编号
日期
客户
销售员
合计
状态
SO006
2014/04/04
上海大众汽车有限公司
李旦
655.20
报价单草稿
655.20
销售
客户
线索
商机
报价单
销售订单
我的任务
合同
售后服务
投诉
服务和支持
款项催收
我的催款
电话访问
通话记录
通话计划
开票
按工时和材料开发票
按订单明细开票
按发货单开票
要续签的合同
产品
产品分类
产品
设置
问卷调查
合同模板
Powered by OpenERP

第1章 概述 Odoo

1.1 系统概述

Odoo 是一个现代化的商业应用套件,使用 AGPL 许可证，并具有客户关系管理（CRM），人力资源，销售，采购，会计，制造，仓库管理，项目管理，以及众多社区模块。它是基于一个模块化，可扩展和直观的快速开发应用程序（RAD）的框架，使用 Python 语言。

OpenObject 功能对象集成- 关系映射（ORM），基于模板的模型 - 视图 - 控制器（MVC）接口，报表生成系统，多国语言，快速构建应用程序：是一个完整的模块化的工具。

Python 是一种解释型、面向对象、动态数据类型的高级程序设计语言，非常适合 RAD 清晰的语法.

相关链接

- **官方下载地址：** http://www.Odoo.com
- **公司网站：** http://www.100china.cn
- **功能及技术文档：** http://doc.Odoo.com
- **社区资源：** http://www.launchpad.net/openobject
- **官方演示地址：** http://demo.Odoo.com
- **定制中国版地址：** http://www.amoserp.com
- **学习 Python：** http://doc.python.org
- **Odoo 电子学习平台：** http://edu.Odoo.com
- **中文社区：** http://www.Odoobbs.com
- **PgAdmin：** http://www.pgadmin.org
- **Postgresql：** http://www.postgresql.org
- **Odoo BBS：** http://www.Odoobbs.com

QQ 群	
Odoo 权限设计器	81740555
Odoo 代码生成器	176403579,343097692
Odoo PO 翻译工具	77839650,326701763
Odoo BOM 设计器	70858940
Odoo 智能浏览器	94666576
Odoo 中文文档开发教程	80748095,249757342(火爆)
Odoo 视频开发教程	364300395,60280211
Odoo 中文文档实施教程	94666546

Odoo 中文实施视频教程	175579111
ODoo 数据专用导入工具	51131926
Odoo or OpenERP 中国群	72091745(火爆)
Odoo AmosERP	86732947
Odoo 课程报名	342115998
Odoo 使用手册读者群	333812183
Odoo 四通框架群	306456635
OpenERP 售后服务群	249941048

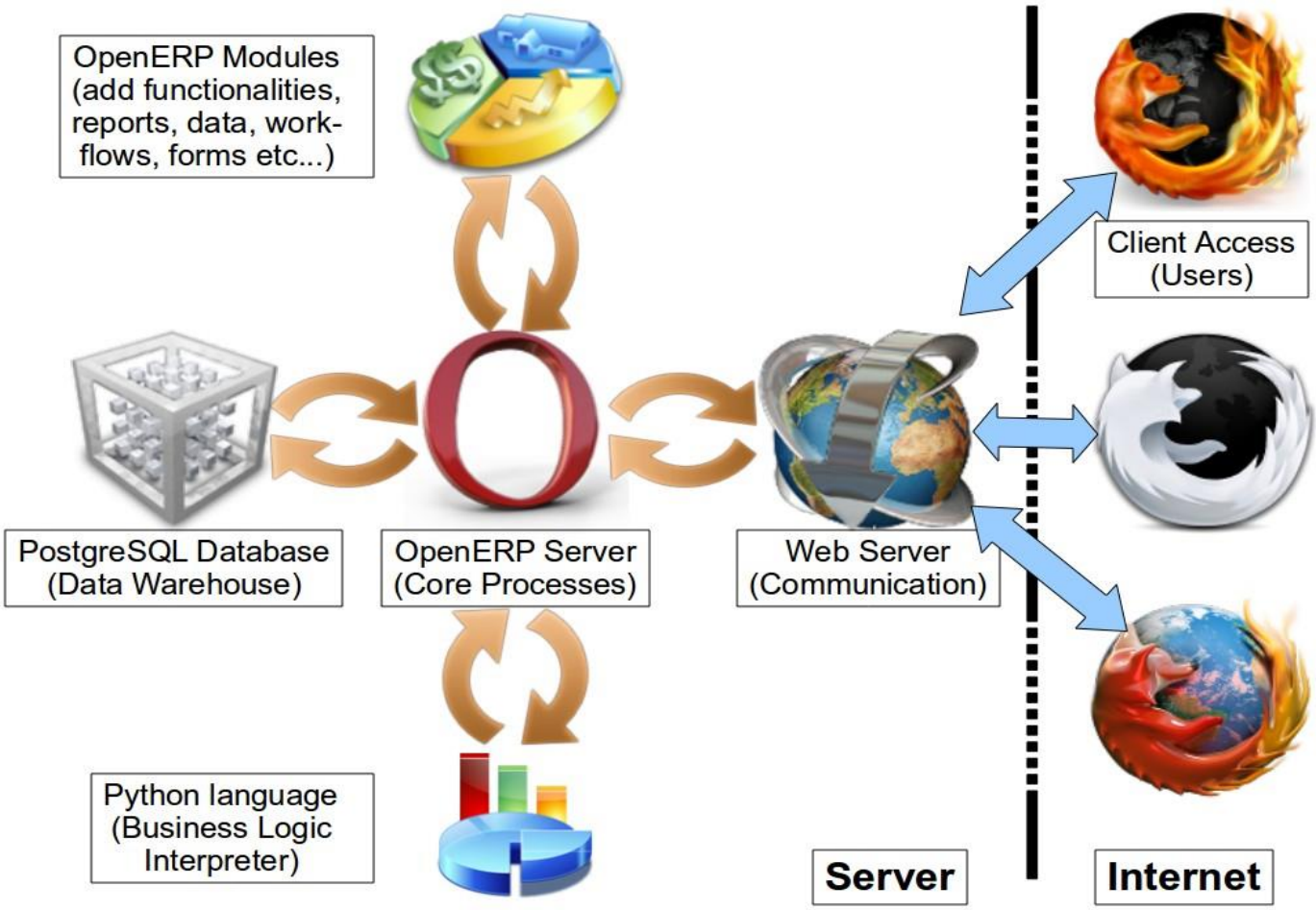
1.2 安装说明

Odoo 是分布式安装包/可安装在大多数平台上.

Note:
安装的程序很可能增加依赖包，即时关注官方的安装说明文档 http://doc.Odoo.com/install
如果是收费客户可以免费获取两个平台按装包（Windows，Linux）

1.3 Odoo 架构

Odoo 使用客户端与服务器模式：客户端使用浏览器运行，连接到服务器上使用 JSON-RPC 协议通过 HTTP（S）。用户也可以很容易地编写和使用 XML-RPC 或 JSON-RPC 连接到服务器。



1.4 安装软件包

Windows	all-in-one installer
Linux	all-in-one packages available for Debian-based (.deb) and RedHat-based(.rpm) distributions
Mac no	all-in-one installer, needs to be installed from source

python 自动安装工具 setuptools(easy_install) 的使用

1. 下载安装 python 安装工具，下载地址：http://pypi.python.org/pypi/setuptools ，可以找到正确的版本进行下载。

CMD 安装：进入解压目录执行 python python.py install 【pypa-setuptools-7a8e47b11640.tar.gz】

2.解压缩后双击 ez_setup.py 进行安装，相关文件将自动安装至当前 python 版本的 scripts 目录下，如：C:\Python27\Scripts 。(或在 cmd 下执行 python ez_setup.py install，即可自动安装 setuptools)。

3.安装 python 模块时，首先 cmd 进入 C:\Python27\Scripts 目录，执行安装命令，如安装 phonenumbers 模块时执行 " easy_install phonenumbers " 命令即可。

以下内容为转载，原文地址：http://bu-choreography.iteye.com/blog/1113059

安装 easy_install 完成之后，如何使用 easy_install 呢，去官网看看吧：--Downloading and Installing a Package

使用一：

根据你想要的安装包名来进行 easy_install，工具会检索网页查询最新版本的包，自动下载、构建和安装

easy_install SQLObject

这办法很简洁并不适合中国国情，由于 GFW 对 python.org 的长期屏蔽，命令行下的 easy_install 根本找不到网址，更无从下载

使用二：

指定网址来更新或安装，多了个参数-f 和用来指定页面的地址 只指定页面地址

easy_install -f http://pythonpaste.org/package_index.html SQLObject

使用三：

只使用网址来 easy_install 安装

easy_install http://example.com/path/to/MyPackage-1.2.3.tgz

使用四：

easy_install 安装下载好的 egg 文件（egg 文件是用 setup tools 打包的压缩文件）

easy_install /my_downloads/OtherPackage-3.2.1-py2.3.egg

使用五：

升级包，有时候你需要更新包的版本

easy_install --upgrade PyProtocols

使用六：

翻*墙后下载安装包并解压，然后 CD 到解压包内运行以下命令即可

easy_install .

参考资料：http://peak.telecommunity.com/DevCenter/EasyInstall#downloading-and-installing-a-package

1.5 源代码安装

源代码托管在 Launchpad 。用 Bazaar 从 Launchpad 获取代码。Bazaar 是一个版本控制系统，帮助跟踪项目的变动历史以及提高协作效率。您需要在 Launchpad 上创建帐户就可以参与 OenERP 开发。请参考的 Launchpad 和 Bazaar 文档，去安装和设置您的开发环境。

官方源码包地址：http://nightly.Odoo.com/7.0/nightly/src/

你还需要安装所需的依赖程序（PostgreSQL 和几个 Python 库 - 见 doc.Odoo.com 文档）。

下载 Linux 下 Odoo

```
$ sudo apt-get install bzr              #下载社区版本
$ bzr branch lp:Odoo                    #按装 Odoo
$ cd Odoo && python ./bzr_set.py  #获取代码并执行安装
```

Note:
Odoo 使用 Python-开发，所以不需编译

1.6 Ubuntu 桌面版 12.04 64 位 安装 Odoo

从 Ubuntu 官网上下载 12.04 Server 的 ISO 文件装好系统

1.7 创建 Ubuntu 用户

首先打开终端，输入：sudo su,会切换到 root 用户下。

输入：sudo adduser username，系统会提示以下信息：

正在添加用户“username”...

正在添加新组“username”(1001)...

正在添加新用户“username”(1001)到组“username”...

创建主目录“/home/username”...

正在从“/etc、skel”复制文件...

输入新的 UNIX 口令：(此处大家注意，不是输入你当前用户的密码，而是输入你要创建新用户的密码)

重新输入新的 UNIX 口令：(再输一次即可)

passwd：已成功更新密码

Changing the user information for username

Enter the new value, or press ENTER for the default

Full Name []: amoserpname (输入新用户的名称)

Room Number []:

Work Phone []:

Home Phone []:

Other []:

这个信息是否正确? [Y/n] y

到了这一步，新用户已经添加成功了

1.8 如何删除 ubuntu 用户？

ubuntu 删除用户同样是在终端下操作的，需要注意的是，如果要删除的用户当前已登陆，是删除不掉的，必须注销掉当前用户切换为另一个用户下，才能删除。举个例子，刚才我新建立了一个用户为 amoserp 的用户，例如我现在用用户 amoserp 登陆了桌面，此时如果我想删除 amoserp 这个用户，是删除不掉的。正确的操作方法是，我注销掉 amoserp，然后使用 root 登陆到桌面，再删除 amoserp 即可。

删除 ubuntu 用户的命令比较容易记：sudo userdel username，例如我想删除 amoserp ，则输入：sudo userdel amoserp，删除成功后，系统无任何提示。

用户间切换：

可以使用：su username,

需要输入口令。

1.8.1 Postgresql 安装

说明：

我是用 root 用户在终端登陆的，如果是非 root 用户，那在命令前需要加上"sudo"

1.8.1.1 第一步：

1.使用 apt-get install 安装

amos@ubuntu:~$ sudo apt-get install postgresql

[代码说明]

安装服务端和命令行客户端 psql。等待一段时间，系统会自动从网上下载下安装文件并完成安装

1.8.1.2 第二步：修改 PostgreSQL 数据库的默认用户 postgres 的密码(注意不是 linux 系统帐号)

2.PostgreSQL 登录(使用 psql 客户端登录)

amos@ubuntu:~ $ sudo -u postgres psql

//其中，sudo -u postgres 是使用 postgres 用户登录的意思

//PostgreSQL 数据默认会创建一个 postgres 的数据库用户作为数据库的管理员，密码是随机的，所以这里

//设定为'postgres'

3.修改 PostgreSQL 登录密码：

postgres=# ALTER USER postgres WITH PASSWORD 'postgres';

//postgres=#为 PostgreSQL 下的命令提示符

4.退出 PostgreSQL psql 客户端

postgres=# \q

[代码说明]

#’和’#'之前的字符是系统提示符，’postgres=#’是 psql 客户端的提示符，红色字符为输入命令（本文其它部分亦如此）；

[功能说明]

PostgreSQL 数据默认会创建一个 postgres 的数据库用户作为数据库的管理员，密码是随机的，我们需要修改为指定的密码，这里设定为’postgres’

1.8.1.3 第三步：修改 linux 系统的 postgres 用户的密码（密码与数据库用户 postgres 的密码相同）

1.删除 PostgreSQL 用户密码

amos@ubuntu:~ $ sudo passwd -d postgres

passwd: password expiry information changed.

//passwd -d 是清空指定用户密码的意思

2.设置 PostgreSQL 用户密码

PostgreSQL 数据默认会创建一个 linux 用户 postgres，通过上面的代码修改密码为'postgres’（这取决于 第二步中的密码，只要与其相同即可）。

现在，我们就可以在数据库服务器上用 postgres 帐号通过 psql 或者 pgAdmin 等等客户端操作数据库了。

amos@ubuntu:~ $ sudo -u postgres passwd

输入新的 UNIX 密码：

重新输入新的 UNIX 密码：

passwd：已成功更新密码

1.8.1.4 第四步：修改 PostgresSQL 数据库配置实现远程访问

amos@ubuntu:~ $ vi /etc/postgresql/9.1/main/postgresql.conf

1.监听任何地址访问，修改连接权限

#listen_addresses = 'localhost' 改为 listen_addresses = '*'

2.启用密码验证

#password_encryption = on 改为 password_encryption = on

3.可访问的用户 ip 段

amos@ubuntu:~ $ vi /etc/postgresql/9.1/main/pg_hba.conf,并在文档末尾加上以下内容

to allow your client visiting postgresql server

host all all 0.0.0.0 0.0.0.0 md5

4.重启 PostgreSQL 数据库

amos@ubuntu:~ $ /etc/init.d/postgresql restart

1.8.1.5 第五步：管理 PostgreSQL 用户和数据库

1.登录 postgre SQL 数据库

amos@ubuntu:~ $ psql -U postgres -h 127.0.0.1

2.创建新用户 amos，但不给建数据库的权限

postgres=$ create user "amos" with password 'openpgpwd nocreatedb;

//注意用户名要用双引号，以区分大小写，密码不用

3.建立数据库，并指定所有者

postgres=$ create database "demo" with owner=" amos"；

4.在外部命令行的管理命令

amos@ubuntu:~ $ -u postgres createuser -D -P test1

//-D 该用户没有创建数据库的权利，-P 提示输入密码,选择管理类型 y/n

amos@ubuntu:~ $ -u postgres createdb -O test1 db1

//-O 设定所有者为 test1

1.8.1.6 第六步：安装 postgresql 数据库 pgAdmin3 客户端管理程序

amos@ubuntu:~ $ sudo apt-get install pgadmin3

如果要在 Ubuntu 的图形界面启动 pgadmin，只需要按下键盘的 windows 键，在搜索中输入 pgadmin，就可以查找到它，点击就可以启动。如果要方便以后使用，可以把它拖到启动器上锁定就行了。

1.8.2 安装 Python 的依赖

这一步很简单，Odoo 不会自动安装所有需要运行的 Python 库。你需要手动安装。

提示：使用键盘上的向上箭头调出以前的命令。然后修改库名。不需要一遍又一遍重输入安装命令

最好安装一个 PIP 可能部分模块装不了就用 PIP

这是 PIP 安装方式 http://www.pip-installer.org/en/latest/installing.html#python-os-support

sudo apt-get install python-pip

然后删除 Ubuntu 的的打包版本 WERKZEUG

sudo apt-get remove python-werkzeug

然后安装最新版本 WERKZEUG

sudo pip install werkzeug

- **sudo apt-get install python-docutils**
- **sudo apt-get install python-gdata**
- **sudo apt-get install python-mako**
- **sudo apt-get install python-dateutil**
- **sudo apt-get install python-feedparser**
- **sudo apt-get install python-lxml**
  https://pypi.python.org/pypi/lxml/2.3#downloads
- **sudo apt-get install python-libxslt1**
- **sudo apt-get install python-ldap**
- **sudo apt-get install python-reportlab**
- **sudo apt-get install python-pybabel**
- **sudo apt-get install python-pychart**
- **sudo apt-get install python-openid**
- **sudo apt-get install python-simplejson**
- **sudo apt-get install python-psycopg2**
  http://initd.org/psycopg/download/
- **sudo apt-get install python-vobject**
- **sudo apt-get install python-tz**
- **sudo apt-get install python-vatnumber**
- **sudo apt-get install python-webdav**
- **sudo apt-get install python-xlwt**
- **sudo apt-get install python-werkzeug**

- **sudo apt-get install python-yaml**
- **sudo apt-get install python-zsi**
- **sudo apt-get install python- win32service**

http://sourceforge.net/projects/pywin32/files/pywin32/Build%20214/pywin32-214.win32-py2.7.exe/download

可尝试在下面地址里查看

http://www.lfd.uci.edu/~gohlke/pythonlibs/

https://pypi.python.org/simple/

1.8.3 下载并安装 Odoo 的软件

第 1 步：下载原码

wget http://nightly.Odoo.com/trunk/nightly/src/Odoo-8.0dev-latest.tar.gz

第 2 步：解压缩 tar 文件

sudo adduser --system --home=/opt/Odoo --group Odoo

sudo chmod 777 /opt

cd /opt/Odoo

sudo tar xvf ~/Odoo-8.0dev-latest.tar.gz

1.8.4 配置 Odoo 的配置文件

第 1 步: 复制配置文件

在安装目录中，我们可以复制 Odoo-server.conf 到/ etc 目录中。

sudo cp /opt/Odoo/server/install/Odoo-server.conf /etc/

sudo chown Odoo: /etc/Odoo-server.conf

sudo chmod 640 /etc/Odoo-server.conf

第 2 步: 打开 Odoo 的配置文件

sudo nano /etc/Odoo-server.conf

第 3 步: 更改配置文件中的密码

查找 db_password=false 修改成 PostGres 安装密码

1.8.5 启动服务器和测试安装

第 1 步: 登录 Odoo 的帐户

sudo su - Odoo -s /bin/bash

第 2 步: 运行命令来启动服务器

/opt/Odoo/server/Odoo-server

第 3 步: 测试 Odoo

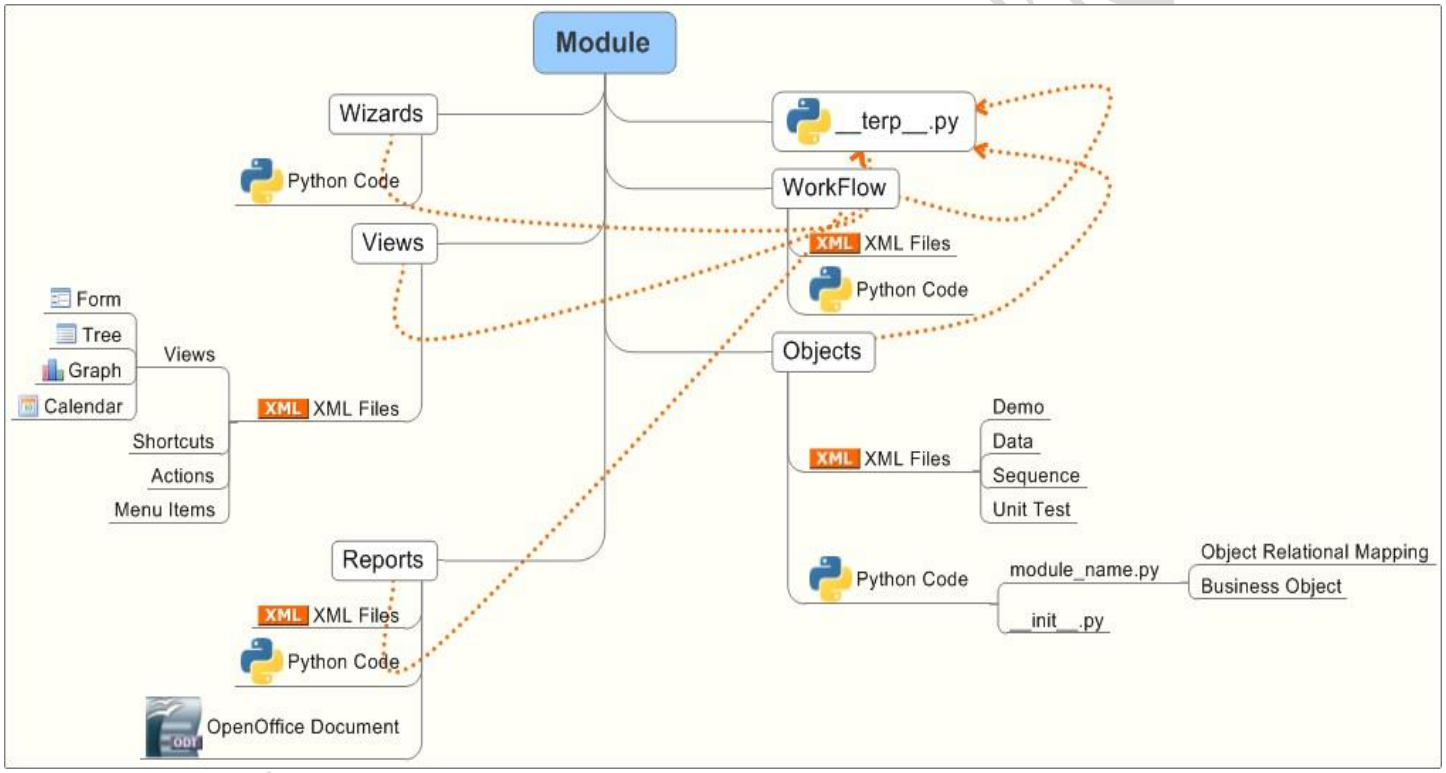
地址： http://127.0.0.1:8069

如果安装成功你应该看到 Odoo 提示你创建你的第一个数据库.

1.9 创建数据库

安装 Odoo 后，启动 Odoo 服务。从 Web 客户端的登录，单击“管理数据库，以创建一个新的数据库（默认超级管理员密码为 admin）。每个数据库都有它自己的模块和配置.

Note:
创建时可选择是否添加演示数据。，也可以在线恢复数据，与备份数据库 http://127.0.0.1:8069/web/database/manager 这是数据库管理地址



```
[options]
; This is the password that allows database operations log_level = debug:
; admin_passwd = admin
admin_passwd = admin
db_host  = 127.0.0.1
db_port = 5432
db_user = oe
db_password = openpgpwd
pg_path= C:\Program Files\PostgreSQL\9.2\bin
addons_path = amos_public,amos_all,amos_tc
```

界面上创建与恢复数据库帐套

windows 下自动备份时需要地址

自己开发的模块可以放在指定文件夹

1.10 版本升级

版本升级

-c setup.cfg --database=SS --update=all

版本更新

-c setup.cfg --database=SS -- init =all

1.11 常规选项

格式:	
--version	显示当前版本号
-h, --help	显示帮助
-c CONFIG, --config=CONFIG	specify alternate config file
-s, --save	save configuration to ~/.terp_serverrc
-v, --verbose	启用调试
--pidfile=PIDFILE	文件服务器的 PID 将存储
--logfile=LOGFILE	在服务器的日志文件将存储
-n INTERFACE, --interface=INTERFACE	指定 TCP IP 地址
-p PORT, --port=PORT	指定的 TCP 端口
--no-xmlrpc	禁用 xmlrpc
-i INIT, --init=INIT	初始化一个模块 使用指定名称 或 all
--without-demo=WITHOUT_DEMO	加载模块演示数据 使用指定名称 或 all
-u UPDATE, --update=UPDATE	更新一个模块 使用指定名称 或 all
--stop-after-init	在初始化停止服务器
--debug	启用调试模式

-S, --secure	启动服务器通过 HTTPS 而不是 HTTP
--smtp=SMTP_SERVER	指定 SMTP 服务器发送邮件
addons_path = addons,saas	指定一个路径

1.12 数据库相关的选项

格式:	
-d DB_NAME, --database=DB_NAME	指定的数据库名称
-r DB_USER, --db_user=DB_USER	指定数据库用户名
-w DB_PASSWORD, --db_password=DB_PASSWORD	指定数据库的密码
--pg_path=PG_PATH	指定的执行 pgAdmin 的可执行文件的路径
--db_host=DB_HOST	指定数据库路径 localhost
--db_port=DB_PORT	指定数据库端口 8069

1.13 Pycharm 启动更新指定模块

格式:
-c openerp-server.conf --stop-after-init --update=amos_report, amos_ addons -d 数据库名

1.14 pycharm 快击键

编辑	搜索/替换	导航	动态模板
Ctrl + Space 基本代码完成（任何类别的名称，方法或可变），取当前界面所有方法	Ctrl + F 查询	Ctrl + N Go to class	Ctrl + Alt + J Surround with Live Template
Ctrl + Alt + Space 类名（任何项目的名称目前进口类独立完成）	F3 查找下一个	Ctrl + Shift + N Go to file	Ctrl + J Insert Live Template
Ctrl + Shift + Enter 完整的语句	Shift + F3 查找上一个	Ctrl + Alt + Shift + N Go to 标志	一般
Ctrl + P 参数信息（在方法调用的参数）	Ctrl + R 替换	Alt + Right/Left 转到下一个/上一个编辑器标签	Alt + #[0-9] Open corresponding tool window
Ctrl + Q 快速文档查询	Ctrl + Shift + F 在路径查找	F12 回到以前的工具窗口	Ctrl + S Save all
Shift + F1 外部文件	Ctrl + Shift + R 更换路径	Esc 转到编辑器（从工具窗口）	Ctrl + Alt + Y Synchronize
Ctrl + mouse over code	运行	Shift + Esc	Ctrl + Shift + F12

业务简介		隐藏主动或最后一个活动窗口	Toggle maximizing editor
Ctrl + F1 在插入符号显示错误或警告的描述	Alt + Shift + F10 选择配置和运行	Ctrl + Shift + F4 关闭活动运行/消息//...标签	Alt + Shift + F Add to Favorites
Alt + Insert 生成的代码...	Alt + Shift + F9 选择的配置和调试	Ctrl + G 转到行	Alt + Shift + I Inspect current file with current profile
Ctrl + O 覆盖方法	Shift + F10 运行	Ctrl + E 最近弹出文件	Ctrl + BackQuote (`) Quick switch current scheme
Ctrl + Alt + T 环绕...	Shift + F9 调试	Ctrl + Alt + Left/Right 前进/后退导航	Ctrl + Alt + S Open Settings dialog
Ctrl + / 注释/取消注释行注释	Ctrl + Shift + F10 运行上下文配置编辑器	Ctrl + Shift + Backspace 导航到最后编辑的位置	Ctrl + Shift + A Find Action
Ctrl + Shift + / 注释/取消注释块注释	Ctrl+Alt+R 运行 manage.py 任务	Alt + F1 在任何视图中选择当前文件或符号	Ctrl + Tab Switch between tabs and tool window
Ctrl + W 选择连续增加的代码块	调试	Ctrl + B or Ctrl + Click 转到声明	
Ctrl + Shift + W 当前选择减少到以前的状态	F8 Step over	Ctrl + Alt + B 去实施（S）	
Ctrl + Shift +]/[直到代码块中选择结束/开始	F7 Step into	Ctrl + Shift + I 打开快速定义查找	
Alt + Enter 显示意图的行动和快速修复	Shift + F8 Step out	Ctrl + Shift + B 转到键入声明	
Ctrl + Alt + L 格式化代码	Alt + F9 Run to cursor	Ctrl + U Go to super-method/super-class	
Ctrl + Alt + O 优化进口	Alt + F8 Evaluate expression	Alt + Up/Down Go to previous/next method	
Ctrl + Alt + I Auto-indent line(s)	Ctrl + Alt + F8 Quick evaluate expression	Ctrl +] / [Move to code block end/start	
Tab / Shift + Tab 缩进/取消缩进选中的行	F9 Resume program	Ctrl + F12 File structure popup	
Ctrl + X or Shift + Delete 剪切当前行或选定块到剪贴板	Ctrl + F8 Toggle breakpoint	Ctrl + H Type hierarchy	
Ctrl + C or Ctrl + Insert 复制当前行或选定块到剪贴板	Ctrl + Shift + F8 View breakpoints	Ctrl + Shift + H Method hierarchy	
Ctrl + V or Shift + Insert 从剪贴板粘贴	使用搜索	Ctrl + Alt + H Call hierarchy	
Ctrl + Shift + V 从最近的缓冲区粘贴...	Alt + F7 / Ctrl + F7	F2 / Shift + F2 Next/previous highlighted error	

	Find usages / Find usages in file		
Ctrl + D 重复当前行或选定的块	Ctrl + Shift + F7 Highlight usages in file	F4 / Ctrl + Enter Edit source / View source	
Ctrl + Y 删除行插入符号	Ctrl + Alt + F7 Show usages	Alt + Home Show navigation bar	
Ctrl + Shift + J 智能线连接	重构	F11 Toggle bookmark	
Ctrl + Enter 智能线分割	F5 Copy	Ctrl + Shift + F11 Toggle bookmark with mnemonic	
Shift + Enter Start new line	F6 Move	Ctrl + #[0-9] Go to numbered bookmark	
Ctrl + Shift + U 切换字插入符号的情况下，或选择块	Alt + Delete Safe Delete	Shift + F11 Show bookmarks	
Ctrl + Delete 删除字结束	Shift + F6 Rename	VCS/本地历史	
Ctrl + Backspace 删除字开始	Ctrl + F6 Change Signature	Ctrl + K Commit project to VCS	
Ctrl + NumPad+/- 展开/折叠代码块	Ctrl + Alt + N Inline	Ctrl + T Update project from VCS	
Ctrl + Shift + NumPad+ 全部展开	Ctrl + Alt + M Extract Method	Alt + Shift + C View recent changes	
Ctrl + Shift + NumPad- 关闭全部	Ctrl + Alt + V Extract Variable	Alt + BackQuote (`) ‘VCS’ quick popup	
Ctrl + F4 关闭活动编辑器“选项卡	Ctrl + Alt + F Extract Field		
	Ctrl + Alt + C Extract Constant		
	Ctrl + Alt + P Extract Parameter		

PyCharm 常用快捷键

Alt+Enter 自动添加包

Ctrl+t SVN 更新

Ctrl+k SVN 提交

Ctrl + / 注释(取消注释)选择的行

Ctrl+Shift+F 高级查找

Ctrl+Enter 补全

Shift + Enter 开始新行

TAB Shift+TAB 缩进/取消缩进所选择的行

Ctrl + Alt + I 自动缩进行

Ctrl + Y 删除当前插入符所在的行

Ctrl + D 复制当前行、或者选择的块

Ctrl + Shift + J 合并行

Ctrl + Shift + V 从最近的缓存区里粘贴

Ctrl + Delete 删除到字符结尾

Ctrl + Backspace 删除到字符的开始

Ctrl + NumPad+/- 展开或者收缩代码块

Ctrl + Shift + NumPad+ 展开所有的代码块

Ctrl + Shift + NumPad- 收缩所有的代码块

第2章 构建 Odoo 模块

Odoo 服务的显着特征之一是对象关系映射 ORM 层。ORM 层在 PostgreSQL 服务上提供了必要和额外功能。数据模型通过 Python 代码描述，Odoo 通过 ROM 创建基础数据库表。 RDBMS 的所有优点，如唯一约束，关系的完整性或高效查询，都通过 Python 灵活实现。例如，在任何模型可添加由 Python 编写的任意限制。 Odoo 也能够带来不同模块的可扩展机制。

在通过 ORM 或直接用原始 SQL 访问数据库之前，先要了解 ORM 的职能。Odoo 的 ORM 可以确保数据干净。例如，一个模块可以响应一个特定表中的数据创建。这种行为只能通过 ORM 进行查询实现。

ORM 包括的其他服务

- 强大的有效性检查，一致性验证
- 提供对象接口，能够设计出更高效的模块（方法，引用，...）
- 用户和组的记录级安全性
- 资源组上的复杂组合
- 继承功能，让新模型更灵活

Odoo 的默认客户端是一个 JavaScript 应用程序，运行在浏览器中，使用 JSON-RPC 服务器通信。

2.1 模块之间的关联图

一个标准模块包括以下几个元素 ：

- **Business object :** 继承 Python 类的 osv.Model，这些由 OpenObject 进行管理持久层。
- **Data :** XML/CSV 文件的源数据（视图和工作流），默认配置数据（模块参数）和演示数据（可选是否加载）。
- **Wizards :** 用于协助用户状态的互动形式，经常用作模块上下文传递。
- **Reports :** RML (XML 格式)。 MAKO 或者 OpenOffice 报表模板，通过数据请求可以生成 HTML, ODT or PDF 报表。

2.2 模块结构

每个模块都包含在自己的目录中或使用插件形式继承原来的模块

Note: 配置 addons 位置
默认情况下读取 server/bin/addons 下的插件，当然你也可以复制一个已知的插手到 addons ,你还可以创建一个连接器到指定目录中 使用 addons_path 将多个文件夹动态加载，文件夹之间使用逗号隔开,注意第一个 addons 是系统默认的 APP 文件夹一定要写，否则会找不到对象。
例：addons_path = addons,amos_public,amos_all,amos_installer,amos_test,amos_sdw,amos_third_addons

init.py

文件是 Python 的导入文件，因为 Odoo 的模块也是一个普通的 Python 模块。该文件用来导入所有其他 python 文件或子模块。

例如，如果一个模块包含一个名为 amos_files.py 的 Python 文件，导入文件格式定义如下：

import amos_files

__openerp__.py 是 Odoo 的一个模块声明。每个模块中都必须包含的文件，模块中定义了几个重要的信息。如下：

```
{
    'name': 'Amos Template', #模块名称
    'summary': '实例开发', #简要描述，在模块列表中显示
    'version': '1.0', #版本
    'category': 'Tools', #分类
    'sequence': 1001, #序号，在本地模块列表中的显示顺序
    'author': 'Amos', #作者
    'website': 'http://www.10china.cn', #网址
    'images': [''], #模块中主要功能截图展示文件
    'depends': ['base'], #依赖模块，即安装本模块时将检查此处定义的模块，如果没安装，将自动一起安装。通常所有模块都要依赖 base 模块。本例工作流要依赖 process 模块，员工及部门经理关系用到 hr 模块
    'data': [
        #模块安装和升级时需要重新加载的 XML 文件，基础数据、权限、工作流、视图、报表等的定义文件通常放在此处。通常权限定义文件放在前面，因为其它文件常引用权限定义数据
    ],
    'demo': [
        #示例数据
    ],
    'test': [
        #测试文件
    ],
    'installable': True, #是否启用安装，通常固定为 True
    'application': True, #是否为应用程序
    'auto_install': False, #建库时是否自动安装
    'description': """
    Odoo 开发实现要领
    ==================================
    通过本模块实例，你可以任意的扩展现有的系统，已适应公司的需求，下面列出各个功能模块名称
    * **菜单** -> **字段** -> **权限** -> **界面** -> **报表-翻译** -> **工作流** -> **邮件** -> **自动动作** -> **版本控制器** -> **审核留言**
    菜单
    ----------------------------------------------------
    如何创建菜单：
    * 创建菜单
    * 排序
```

* 关联触发事件

字段类型:

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

* selection, function, related

* *relational*: one2one one2many many2one Many2many

* 命名空间 from osv import fields

模块开发各个要领:

---

* 权限：权限组生成，规则关联，隐藏字段

* 界面：界面开发，列表，表单，搜索，继承，统计图，甘特图，日历，排序，按钮调用

* 报表开发，报表继承，生成样式（pdf,doc,html）,套打

* 工作流：自定义工作流

* 翻译：菜单翻译，字段翻译，Help 字段翻译，界面翻译

* 邮件：通过模板发邮件

* 自动动作：创建一个定时器自动执行相关提醒动作

* 版本：对单据的历史进行保存（目前只保存明细集）

* 审核：审核时弹出对话框，可留言

""", #模块详细描述，支持简单的富文本格式化显示，使用====，*号等标记

}

__openerp__ 名称也可以用户自己定义：

opener/tools/config.py

def _is_addons_path(self, path): 行约 550

练习 1 - 模块创建
创建一个空的 Odoo 模块，并按装在 Odoo 系统中

2.3 XML 文件

模块目录中的 XML 文件用来在模块安装或更新时更新数据。它们被用于多种用途，我们可以举出：

- 初始化和声明演示数据
- 声明视图
- 声明报表
- 声明工作流

Odoo 的 XML 文件的通常结构比 XML 数据序列化更详细。看这里 如果你有兴趣了解更多关于初始化和定义示范数据的 XML 文件。以下部分仅仅是些特定的 XML 声明： actions, menu entries, reports, wizards and workflows 。

使用 XML 文件，可以在 Odoo 对象对应的 PostgreSQL 表中插入或更新数据。 Odoo XML 文件的结构一般如下：

2.3.1 插入默认值

```
<?xml version="1.0"?>
 <Odoo>
    <data>
            <record model="model.name_1" id="id_name_1">
                <field name="field1"> "field1 content" </field>
                <field name="field2"> "field2 content" </field>
                (...)
            </record>
            <record model="model.name_1" id="id_name_2">
                <field name="field1"> "field1 content" </field>
                <field name="field2"> "field2 content" </field>
                (...)
            </record>
   </data>
</Odoo>
```

2.3.2 标签

新添加的数据会有个记录标记。一个强制属性：模型。模型是插入的对象名称。标签记录，也可带一个可选属性：ID。如果有这个属性，这个变量名可在在同一个文件的后面作为资源 ID 被引用。

一个记录标签可以包含多个字段标签。可以表示记录的字段值。如果未指定字段，将使用默认值。

字段标签的属性有以下几种：

name : mandatory 名称：强制

eval : optional 计算表达式：可选

Ref ： 引用

model ：对象

例子

```
<record model="res.company" id="main_company">
    <field name="name">Tiny sprl</field>
    <field name="partner_id" ref="main_partner"/>
    <field name="currency_id" ref="EUR"/>
</record>
```

```
    <record model="res.users" id="user_admin">

        <field name="login">admin</field>

        <field name="password">admin</field>

        <field name="name">Administrator</field>

        <field name="signature">Administrator</field>

        <field name="action_id" ref="action_menu_admin"/>

        <field name="menu_id" ref="action_menu_admin"/>

        <field name="address_id" ref="main_address"/>

        <field name="groups_id" eval="[(6,0,[group_admin])]"/>

        <field name="company_id" ref="main_company"/>

    </record>
```

这里定义了 admin 用户：

- 登录名，密码等字段值是明文。
- ref 属性填写的记录之间的关系：

```
    <field name="company_id" ref="main_company"/>
```

字段 company_id 是 一个与用户与公司对象的多对一关系，main_company 是关系的关联的 id。

表达式属性可以写一些 Python 代码：这里 groups_id 字段是一个 many2many 中关系类型。对于这样一个字段，"[(6,0,[group_admin])]" 是指：删除所有与当前用户关联的组，并使用[group_admin]作为新的用户组（group_admin 是另一个记录的 id）。

搜索属性在不知道 xml id 的情况下，找到关联的记录。因此，可以指定寻找记录的搜索条件。搜索条件是一个为预定义的搜索方式，是具有相同的形式的元组列表。如果搜索的结果有多个，将任意选择一个（第一个）

```
<field name="partner_id" search="[]" model="res.partner"/>
```

2.3.3 函数标签

一个函数标签可包括其他一些函数标签

model：mandatory 模型：必填

name : mandatory 名称：必填函数名

eval 计算表达式

用来计算出的将被调用函数的参数列表，不含 cr 和 uid

例子：

<function model="ir.ui.menu" name="search" eval="[[('name','=','Operations ')]]"/>

2.4 对象 - ORM

Odoo 核心是一个完整的 ORM 对象关系映射层，开发人员不必编写基本的 SQL 语句。业务对象通过继承 osv.Model 类来渲染。ORM 的特点是使用预定义的属性指定一个业务对象。（注意：osv.Model 等于 osv.osv），因此，你可以使用 Odoo 的对象接口（ORM）查询对象，或直接使用 SQL 语句查询对象。但在 PostgreSQL 数据库中直接写入或读取很危险，比如你会跳过约束检查，或工作流修改的重要步骤。

预定义的 osv.osv 业务对象的属性	
_name (required)	对象的唯一标识符，必须是全局唯一。这个标识符用于存取对象，其格式通常是"ModuleName.ClassName",对应的，系统会字段创建数据库表"ModuleName_ClassName" **快击键：** **格式：** _name = "amos.template"
_columns (required)	字典{字段名→字段声明} **快击键：** **格式：** _columns = {}
	提供字段的默认值与可以使用函数

_defaults	**快击键：**
	格式：
	_defaults = {
	# 'name': lambda obj, cr, uid, context: obj.pool.get('ir.sequence').get(cr, uid, 'amos.template'),
	'name': lambda obj, cr, uid, context: '/',
	'date': lambda *a: time.strftime('%Y-%m-%d'),
	'datetime': lambda *a: time.strftime('%Y-%m-%d %H:%M:%S'),
	'quantity': lambda *a: int(datetime.datetime.now().strftime('%U')) + 1,
	'state': lambda *a: 'draft',
	'user_id': lambda cr, uid, id, c={}: id,
	'department_id': _default_get,
	'manager': _verify_get,
	'active': lambda *a: True,
	'company_id': lambda s, cr, uid, c: s.pool.get('res.company')._company_default_get(cr, uid, 'hr.employee',
	context=c),
	}

_auto	是否自动创建对象对应的 Table，缺省值为: True。当安装或升级模块时，Odoo 会自动在数据库中为模块中定义的每个对象创建相应的 Table。当这个属性设为 False 时，Odoo 不会自动创建 Table，这通常表示数据库表已经存在。例如，当对象是从数据库视图（View）中读取数据时，通常设为 False。
_inherit	继承产品对象，给产品对象添加字段或方法，不需要设置 _name、_table 等属性　　注意：当继承后的子类不定义 _name 属性，则相当于在父类中增加字段和方法，并不创建新对象 当继承后的子类重新定义 _name 属性，则创建一个新的对象，新对象拥有父类中所有的字段和方法，父类不受任何影响。 **快击键：** **格式：** _inherit = "product.product"

_inherits	继承产品模版对象，创建新的产品对象，产品对象与产品对象之间建立多对一关联关系，产品模版中的字段可以等同产品中的字段一样使用 注意：相当于多重集成。子类通过 _inherits 中定义的字段和各个父类关联，子类不拥有父类的字段，但可以直接操作父类的所有字段和方法。 多重继承，一般用于继承抽象类，调用类中的方法，不创建字段？。如 mail.thread 为邮件对象，用于发送消息。 **快击键：** **格式：** _inherits = {'product.template': 'product_tmpl_id'} _inherit = ["calendar.event", "mail.thread", "ir.needaction_mixin"]

_constraints	对象约束，一般用于业务逻辑复杂，无法通过创建数据库约束实现的情况
	#检测同一时间段内是否存在相同的请假单，False 是存在，不允许创建
	快击键：
	格式：
	def _check_date(self, cr, uid, ids):
	for rec in self.browse(cr, uid, ids):
	search_ids = self.search(cr, uid, [('date_from', '<=', rec.date_to), ('date_to', '>=', rec.date_from), ('employee_id', '=', rec.employee_id.id), ('id', '<>', rec.id)])
	if search_ids:
	return False
	return True
	_constraints = [
	(_check_date, u'您在相同的时间段内不允许创建多张请假单!', [u'起始日期',u'结束日期']),
	]

_sql_constraints	数据库约束，最底层级别的约束，模块安装后对象将在 PostgreSQL 对应的表中创建约束 **快击键：** **格式：** _sql_constraints = [('date_check', "CHECK (date_from <= date_to)", u"开始日期必须小于结束日期."), ('days_check', "CHECK (days > 0)", u"请假天数必须大于 0 ."),]

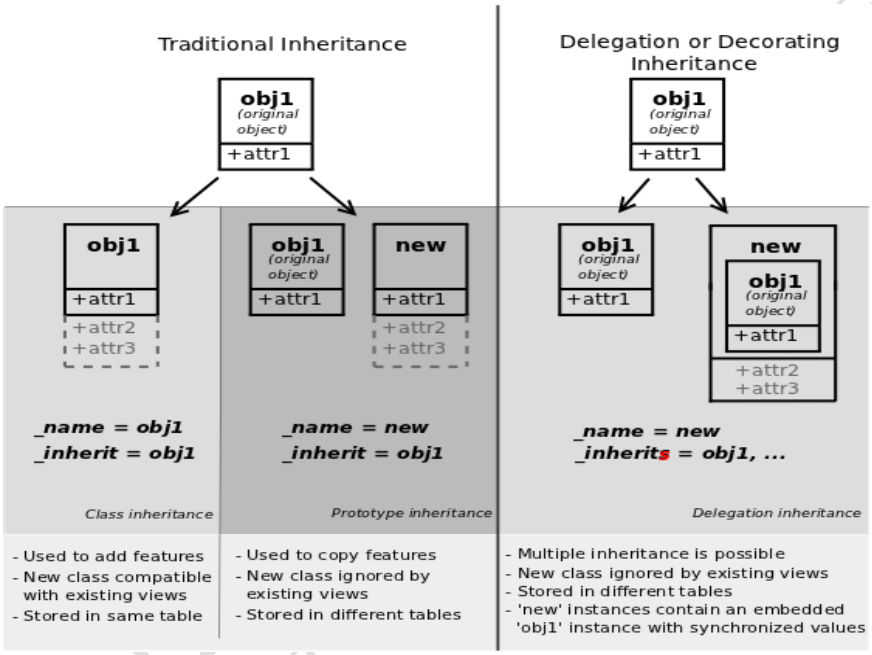
_log_access	是否自动在对应的数据表中增加 create_uid, create_date, write_uid, write_date 四个字段，缺省值为 True，即字段增加。这四个字段分布记录 record 的创建人，创建日期，修改人，修改日期。这四个字段值可以用对象的方法（perm_read）读取 **快击键：** **格式：**
	
_order	定义 search()和 read()方法的结果记录的排序规则，和 SQL 语句中的 order 类似，缺省值是 id,即按 id 升序排序 _order = "name" _order = "date_order desc" **快击键：** **格式：**
_rec_name	标识 record name 的字段。缺省情况（name_get 没被重载的话）方法 name_get()返回本字段值。_rec_name 通常用于记录的显示，例如，销售订单中包含业务伙伴，当在销售订单上显示业务伙伴时，系统缺省的是显示业务伙伴记录的_rec_name **快击键：** **格式：**
_sql	SQL 代码来创建此对象的表/视图（如果_AUTO 是假） - SQL 执行在 init（）方法可以取代 **快击键：** **格式：**

_table	取代 SQL 表名（默认是：_name 点'。'下划线“_”）
init	对象初始化数据 def init(self, cr): 安装对象所属模块时初始化对象数据 示例代码见产品： class product_product(osv.osv): **快击键：** **格式：**
_sequence	数据库表的 id 字段的序列采集器，缺省值为: None。Odoo 创建数据库表时，会自动增加 id 字段作为主键，并自动为该表创建一个序列（名字通常是“表名_id_seq”）作为 id 字段值的采集器。如果想使用数据库中已有的序列器，则在此处定义序列器名 **快击键：** **格式：**

注意事项：

- **_auto** 当_auto 的值为“False"时，OE 不会自动在数据库中创建相应的表，开发者可以在对应类的 init()方法中定义表或视图的 SQL。这一般应用在 报表所对应的数据对象中，因为报表的数据对象往往是“视图”，所以我们可以在 init()方法中创建所需的数据库视图 SQL 即可。
- **_columns** OE 并不一定为每个定义项创建数据库字段，比如：selection, reference, one2many, function 等字段是不会有对应的数据库字段的。
- **_name** 当使用_inherit 时可以与被继承的类的_name 一致，_name 一致表示不创建新的数据库表，而直接在原表上修改
- **_rec_name** 缺省的情况下，name_get 方法使用表中的‘name'字段，所以当你定义了一个实体类 B，并且实体类 B 中即没有“name“字段也没有为 _rec_name 特别指定一个字段， 那么当 OE 调用 name_get 方法时（比如实体类 A 有一个 many2one 字段 b 指向实体类 B，显示 A.b 的值时就会调用类 B 的 name_get 方法） 就会报“未定义 name 字段“的错误
- **_inherits** 另外还可以这样理解_inherits，_inherits 又被称为实例继承，就是新继承的数据对象不但继承被继承对象的属性和方法同时也继承了数据实 例，即表中的记录。比如 product 类不但继承 product_template 的属性和方法，而且这两个表的数据也是同步的。
- **defaults** 在 V6 中字典的值可以不是函数，就比如在 V5 中我们必须这样来定义：_defaults= { 'state' : lambda *a: 'draft'} 而在 V6 中可以这样来：_defaults = {'state' : 'draft'}

2.4.1 对象的扩展

继承有两种方式。这两种方法的结果都是一个新的数据类，拥有父类字段和方法，以及额外的字段和方法，但它们在编程后果上有很大不同。

当例 1 创建新类“custom_material”时，可以被操作“network.material”的视图或树 “看到或使用”，这与例 2 完全不同。这是由于新子类基于的表（other.material），不能够被先前的 “network.material”的视图或树木识别。

例 1:

```
class custom_material(osv.osv):
    _name = 'network.material'
    _inherit = 'network.material'
    _columns = {
        'manuf_warranty': fields.boolean('Manufacturer warranty?'),     }
    _defaults = {
        'manuf_warranty': lambda *a: False,
    }     custom_material()
```

Tip 提醒

Notice 注意

_name == _inherit

在这个例子中，'custom_material'将在对象' network.material'上增加一个新

字段'manuf_warranty'。这个类的新实例，对于能够识别父类表'network.material' 的视图或树来说，也将是可识别的。

在面向对象设计中 inheritancy 通常被称为“类继承”。子类继承数据（字段）和他的父母的行为（函数）

例 2:

```
class other_material(osv.osv):
 _name = 'other.material'
    _inherit = 'network.material'
    _columns = {
        'manuf_warranty': fields.boolean('Manufacturer warranty?'),     }
    _defaults = {
        'manuf_warranty': lambda *a: False,
    }     other_material()
```

Tip 提醒

Notice 注意

_name != _inherit

在这个例子中，“other_material'拥有所有'network.material'的字段，此外，它有一个新的字段'manuf_warranty'。所有这些字段都是表'other.material'的一部分。这个类的新实例，在超类'network.material'表上的视图或树不能识别。

这种类型的 inheritancy 被称为“原型继承”（如 JavaScript），因为新创建的子类从指定的父类（原型）“拷贝”了所有字段。子类继承数据（字段）和他的父母的行为（方法）

2.4.2 委托继承

语法：

```
class tiny_object(osv.osv)
    _name = 'tiny.object'
    _table = 'tiny_object'
    _inherits = {
        'tiny.object_a': 'object_a_id',
        'tiny.object_b': 'object_b_id',
        ... ,
        'tiny.object_n': 'object_n_id'
    }
    (...)
```

对象'tiny.object 从 n 个对象'tiny.object_a', ..., 'tiny.object_n'.继承了所有列和所有的方法

要继承多个表，该技术包括继承对象表 tiny_object 每添加一列，这个列将存储一个外键（另一个表中的 ID）。 值 'object_a_id' 'object_b_id' ... 'object_n_id' 是 string 类型，并确定从'tiny.object_a', ..., 'tiny.object_n'外键列标题已被存储 ，这种继承机制通常被称为“实例继承”或“值继承”。一个资源（实例）有其父母的 VALUES。

2.4.3 ORM 字段类型

在 server/bin/osv/fields.py 里定义继承字段方法

一个字段一具多少可用属性呢
opener/osv/fields.py def __init__(self, string='unknown', required=False, readonly=False, domain=None, context=None, states=None, priority=0, change_default=False, size=None, ondelete=None, translate=False, select=False, manual=False, **args): 约 87 行

解释	
self.states = states or {}	定义字段在单据什么状态下只读与可编辑 格式：states={'done':[('readonly',True)] }
self.string = string	字段名称
self.readonly = readonly	字段只读属性 格式：readonly =True 也可使用 readonly =1
self.required = required	字段必填属性格式：required =True 也可使用 required =1

self.size = size	如果是文本字段 可以指定长度 如果不指定
self.help = args.get('help', '')	
self.priority = priority	
self.change_default= change_default	
self.ondelete = ondelete.lower() if ondelete else None # defaults to 'set null' in ORM	
self.translate = translate	
self._domain = domain	
self._context = context	
self.write = False	
self.read = False	
self.select = select	
self.manual = manual	
self.selectable = True	
self.group_operator = args.get('group_operator', False)	
self.groups = False # CSV list of ext IDs of groups that can access this field	
self.deprecated = False # Optional deprecation warning	

ODOO 所有的数据块都是通过“对象”访问的。作为一个例子，用 res.partner 的对象来访问贸易伙伴有关数据，用 account.invoice 对象访问关于发票的数据..

请注意，每一个资源类型都对应一个对象，而不是每资源一个对象。因此，我们有一个 res.partner 对象来管理所有的合作伙伴，而不是一个的 res.partner 对象对应一个伙伴。如果我们谈论的是“面向对象”的条款，我们也可以说，每一级有一个对象。

直接后果是对象的所有方法都有一个共同的参数：“ids”参数。这是标记每个资源（例如，每个合作伙伴）的方法。准确地说，这个参数包含一个必须应用该方法的资源 ids 列表。

要定义一个新对象，你必须定义一个新 Python 类，然后实例化它。这个类必须继承自 OSV 模块中的 OSV 类。

对象定义的是固定的格式

实例类

```
class name_of_the_object(osv. osv_memory):
        _name = '对象名'
        _columns = { 定义字段 }
```

以下介绍字段类型与可用的常规可选参数

ORM 的字段类型对象包含 3 种类型的字段：基础类型,复杂类型,关系类型。

基础类型：char, text, boolean, integer, float, date, time, datetime, binary

复杂类型：selection, function, related

关系类型：one2one, one2many, many2one, many2many

boolean: 布尔型(true, false)

integer: 整数

float: 浮点型如 'rate' : fields.float(u'金额',digits=(12,6)), digits 定义整数部分和小数部分的位数

char: 字符型，size 属性定义字符串长度

text: 文本型，没有长度限制

date: 日期型

datetime: 日期时间型

binary: 二进制型,可存图片，文档

options ： options='{"no_open": True}' 不会弹出窗口 就是 many2oneXM 界面不再显示超链

options="{'reload_on_button': true}" one2many 首先加载记录 然后调用按钮动作重新加载

options="{' always_reload': true}" one2many 每次插入数据时重新载入记录

function: 函数型，该类型的字段值由函数计算而得，不存储在数据表中。其定义格式为：

fields.function(fnct, arg=None, fnct_inv=None, fnct_inv_arg=None, type='float', fnct_search=None, obj=None, method=False, store=True)

· ***type*** 是函数返回值的类型。【char，date，integer】等 Odoo 常用类型

· ***method*** 为 True 表示本字段的函数是对象的一个方法，为 False 表示是全局函数，不是对象的方法。如果 method=True，obj 指定 method 的对象。

· ***fcnt*** 是函数或方法，用于计算字段值。如果 method = true，表示 fcnt 是对象的方法，其格式如下：def fnct(self, cr, uid, ids, field_name, args, context)，否则，其格式如下：def fnct(cr, table, ids, field_name, args, context)。ids 是系统传进来的当前存取的 record id。field_name 是本字段名，当一个函数用于多个函数字段类型时，本参数可区分字段。args 是'arg=None'传进来的参数。

· ***fcnt_inv*** 是用于写本字段的函数或方法。如果 method = true，其格式是：def fcnt_inv(self, cr, uid, ids, field_name, field_value, args, context)，否则格式为：def fcnt_inv(cr, table, ids, field_name, field_value, args, context)

· ***fcnt_search*** 定义该字段的搜索行为。如果 method = true，其格式为：def fcnt_search(self, cr, uid, obj, field_name, args)，否则格式为：def fcnt_search(cr, uid, obj, field_name, args)

· ***store*** 表示是否希望在数据库中存储本字段值，缺省值为 False。不过 store 还有一个增强形式，格式为 store={'object_name':(function_name,['field_name1','field_name2'],priority)} ，其含义是，如果对象'object_name'的字段['field_name1','field_name2']发生任何改变，系统将调用函数 function_name，函数的返回结果将作为参数(arg)传送给本字段的主函数，即 fnct。

selection: 下拉框字段。定义一个下拉框，允许用户选择值。如：'state': fields.selection((('n','Unconfirmed'),('c','Confirmed')),'State', required=True)，这表示 state 字段有两个选项('n','Unconfirmed')和('c','Confirmed')。

one2one: 一对一关系，格式为：fields.one2one(关联对象 Name, 字段显示名, ...)。在 V5.0 以后的版本中不建议使用，而是用 many2one 替代。

many2one: 多对一关系，格式为：fields.many2one(关联对象 Name, 字段显示名, ...)。可选参数有：ondelete，可选值为"cascade"和"null"，缺省值为"null"，表示 one 端的 record 被删除后，many 端的 record 是否级联删除。

one2many: 一对多关系，格式为：fields.one2many(关联对象 Name, 关联字段, 字段显示名, ...),例：'address': fields.one2many('res.partner.address', 'partner_id', 'Contacts')。

many2many: 多对多关系。例如：'category_id':fields.many2many('res.partner.category','res_partner_category_rel','partner_id','category_id','Categories'), 表示以多对多关系关联到对象 res.partner.category，关联表为'res_partner_category_rel'，关联字段为'partner_id'和'category_id'。当定义上述字段时，Odoo 会自动创建关联表为'res_partner_category_rel'，它含有关联字段'partner_id'和'category_id'。

reference: 引用型，格式为：fields.reference(字段名, selection, size, ...)。其中 selection 是: 1)返回 tuple 列表的函数，或者 2)表征该字段引用哪个对象(or model)的 tuples 列表。reference 字段在数据库表中的存储形式是（对象名，ID），如(product.product,3)表示引用对象 product.product（数据表 product_product）中 id=3 的数据。

reference 的例子：

```
def _links_get(self, cr, uid):
    cr.execute('select object,name from res_request_link order by priority')
    return cr.fetchall()
...
'ref':fields.reference('Document Ref 2', selection=_links_get, size=128),
...
```

上例表示，字段 ref 可以引用哪些对象类型的 resource，可引用的对象类型从下拉框选择。下拉框的选项由函数_links_get 返回，是(object,name)对的列表，如[("product.product","Product"), ("account.invoice","Invoice"), ("stock.production.lot","Production Lot")] 。

related: 关联字段，表示本字段引用关联表中的某字段。格式为：fields.related(关系字段,引用字段,type, relation, string, ...)，关系字段是本对象的某字段（通常是 one2many or many2many），引用字段是通过关系字段关联的数据表的字段，type 是引用字段的类型，如果 type 是 many2one or many2many, relation 指明关联表。例子如下：

'address': fields.one2many('res.partner.address', 'partner_id', 'Contacts'),

'city':fields.related('address','city',type='char', string='City'),

'country':fields.related('address','country_id',type='many2one', relation='res.country', string='Country'),

这里，city 引用 address 的 city 字段，country 引用 address 的 country 对象。在 address 的关联对象 res.partner.address 中，country_id 是 many2one 类型的字段，所以 type='many2one', relation='res.country'。

property: 属性字段，下面以具体例子解说 property 字段类型。

'property_product_pricelist': fields.property('product.pricelist', type='many2one', relation='product.pricelist',string="Sale Pricelist", method=True, view_load=True, group_name="Pricelists Properties")

这个例子表示，本对象通过字段'property_product_pricelist'多对一(type='many2one')关联到对象 product.pricelist(relation='product.pricelist')。和 many2one 字段类型不同的是，many2one 字段会在本对象中创建数据表字段'property_product_pricelist'，property 字段类型不会创建数据表字段'property_product_pricelist'。property 字段类型会从数据表 ir.property 中查找 name='property_product_pricelist'(即字段定义中的'product.pricelist'加上前缀 property，并将"."替换成"_"作为 name)且 company_id 和本对象相同的记录，从该记录的 value 字段(value 字段类型为 reference)查得关联记录，如(product.pricelist,1)，表示本对象的 resource 多对一关联到对象 product.pricelist 的 id=1 的记录。也就是说，property 字段类型通过 ir.property 间接多对一关联到别的对象。

property 字段类型基本上和 many2one 字段类型相同，但是有两种情况优于 many2one 字段。其一是，例如，当有多条记录通过 ir.property 的 name='property_product_pricelist'的记录关联到记录(product.pricelist,1)，此时，如果希望将所有关联关系都改成关联到记录(product.pricelist,2)。如果是 many2one 类型，不写代码，很难完成此任务，是 property 字段的话，只要将 ir.property 中的 value 值(product.pricelist,1)改成(product.pricelist,2)，则所有关联关系都变了。修改 ir.property 的 value 值可以在系统管理下的菜单 Configuration --> Properties 中修改。其二是，例如，同一业务伙伴，但希望 A 公司的用户进来看到的该业务伙伴价格表为 pricelistA，B 公司的用户进来看到的该业务伙伴价格表为 pricelistB，则 many2one 类型达不到该效果。property 类型通过 ir.property 中的记录关联时加上了 company_id 的条件，因此可以使得不同公司的员工进来看到不同的关联记录。

由于 property 类型通过 ir.property 关联，因此，每个 property 类型的字段都必须在 ir.property 中有一条关联记录。这可以在**安装时导入该条记录，参考代码如下：**

```
<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>
```

2.4.4 字段定义的参数

字段定义中可用的参数有， change_default，readonly，required，states，string，translate，size，priority，domain，invisible，context，selection。

password="True" 密码星号显示

nolabel="1" 隐藏标签

attrs 属性 可以定义多条件字段只读，是否显示

digits 直接格式化浮点字段

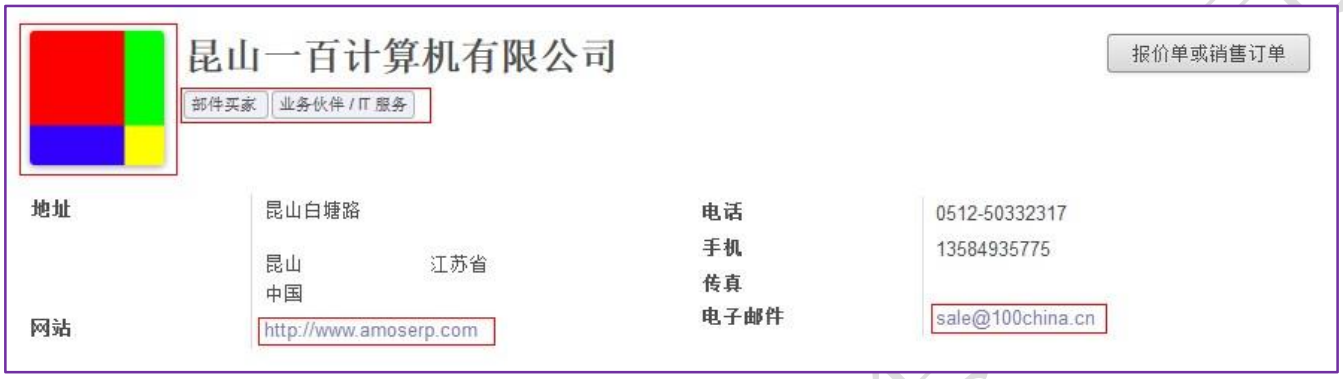
<field digits="(14, 3)" name="volume" attrs="{'readonly':[('type','=','service')]}"/>

default_focus 新开窗口光标位置 <field name="name" default_focus=" 1" />

Widget：有多种部件显示格式 widget="one2many_list"

one2one_list，one2many_list，many2one_list，many2many，url，email，image，float_time，reference, many2many_tags, selection，handle

例：



widget="url" 三种使用方法

第一种：<field name="field_name" widget="url"/>

第二种：form view 使用

第三种：使用按钮

```
return {
    'type': 'ir.actions.act_url',
    'http://www.100china.cn': drawing_url,
   'nodestroy': True,
   'target': 'new' }
```

Sequence 字段设置了 int 内置可以自动拖放排序

widget="handle" 可以指定 int 型字段进行拖放排序

select="1" 默认搜索（定义的 name），select="2"高级搜索

select=True 将在数据库中创建索引来优化当前字段，过滤（与数据库索引），主要用在搜索视图

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)，

'zip': fields.char('Zip', change_default=True, size=24),

这个例子中，可以根据 zip 的值设定其它字段的缺省值，例如，可以通过程序代码，如果 zip 为 200000 则 city 设为“上海”，如果 zip 为 100000 则 city 为“北京”。

无论用户是否可以在定义依赖于这个字段的字段的默认值。这些默认值需要在 ir.values 表里定义

ondelete: 如何处理删除相关的记录。允许的值是：'restrict', 'no action', 'cascade', 'set null', and 'set default'

readonly: 本字段是否只读，缺省值：False

required: 本字段是否必须的，缺省值：False

states: 定义特定 state 才生效的属性，格式为：{'name_of_the_state': list_of_attributes}，其中 list_of_attributes 是形如[('name_of_attribute', value), ...]的 tuples 列表。例子(参见 account.transfer)：

'partner_id': fields.many2one('res.partner', 'Partner', states={'posted':[('readonly',True)]}),

string: 字段显示名，任意字符串

translate: 本字段值（不是字段的显示名）是否可翻译，缺省值：False

size: 字段长度

priority: Not used? 优先级 ，没有用？

domain: 域条件，缺省值：[]。在 many2many 和 many2one 类型中，字段值是关联表的 id，域条件用于过滤关联表的

record：例子：'partner_ref': fields.related("partner_id", "ref", type="char", string="Partner ref.", readonly=True),

'default_credit_account_id': fields.many2one('account.account', 'Default Credit Account', domain="[('type','!=','view')]"),

本例表示，本字段关联到对象('account.account')中的，type 不是'view'的 record

invisible: 本字段是否可见，即是否在界面上显示本字段，缺省值 True

selection: 只用于 reference 字段类型，参见前文 reference 的说明

另外一个问题就是 function 的 store=true 有没有意义呢，不是每次读取这个 function 就会执行里面的方法，那么还储存在数据库里面有什么作用？

我的理解是 store=true 可以将每次函数计算的结果存储到数据库中，这样有两个好处，一是便于 DBA 从数据库查看数据；二是非 Odoo 的软件可以直接访问数据库共享数据。另外补充一点，fcnt_inv 不限于写入本对象的数据表，更经常的情形是写入别的对象，甚至同时写入多个对象的数据表。

<field name="value" eval="'product.pricelist,'+str(list0)"/>

前文说过，value 字段类型为 reference，即形如(product.pricelist,1)的值。这一行就是计算 value 的值，形如 (product.pricelist,list0)，list0（经导入后其值是数字）是引用的 product.pricelist 中的一条记录的 id，如下。

```
<record id="list0" model="product.pricelist">
    <field name="name">Default Purchase Pricelist</field>
    <field name="type">purchase</field>
</record>
```

也就是说，在<field name="value" eval="'product.pricelist,'+str(list0)"/>之前，必须先有上述导入 id="list0"的代码行。

每个 Odoo 的对象都有一些预定义方法，这些方法定义在基类 osv.osv 中。这些预定义方法有:

更多属性参数查看官方文档：https://doc.Odoo.com/6.1/zh_CN/developer/03_modules_3/

on_change:

example,on_change="onchange_shop_id(shop_id)".

视图中的 on_change 属性默认值。这在字段值客户端中变化时，将启动服务器上的定义的函数，当在客户端领域的变化。

2.4.5 基本方法

说明：【create, search, read, browse, write, unlink】。

def create(self, cr, uid, vals, context={})

def search(self, cr, uid, args, offset=0, limit=2000)

def read(self, cr, uid, ids, fields=None, context={})

def browse(self, cr, uid, select, offset=0, limit=2000)

def write(self, cr, uid, ids, vals, context={})

def unlink(self, cr, uid, ids)

2.4.6 缺省值存取方法

说明：default_get, default_set。

def default_get(self, cr, uid, fields, form=None, reference=None)

def default_set(self, cr, uid, field, value, for_user=False)

2.4.7 特殊字段操作方法

说明：perm_read, perm_write

def perm_read(self, cr, uid, ids)

def perm_write(self, cr, uid, ids, fields)

2.4.8 字段(fields)和视图(views)操作方法

说明：fields_get, distinct_field_get, fields_view_get

def fields_get(self, cr, uid, fields = None, context={})

def fields_view_get(self, cr, uid, view_id=None, view_type='form',context={})

def distinct_field_get(self, cr, uid, field, value, args=[], offset=0,limit=2000)

2.4.9 记录名字存取方法

说明：name_get, name_search

def name_get(self, cr, uid, ids, context={})

def name_search(self, cr, uid, name='', args=[], operator='ilike',context={})

#同时按手机、电话、名称模糊查找选择客户

```
    def name_search(self, cr, user, name, args=None, operator='ilike', context=None, limit=100):
        if not args:
            args = []
        args = args[:]
        ids = []
        if name:
            ids = self.search(cr, user, [('mobile', 'ilike', name)]+args, limit=limit, context=context)
            if not ids:
                ids = self.search(cr, user, [('phone', 'ilike', name)]+ args, limit=limit, context=context)
            if not ids:
                ids = self.search(cr, user, [('name', operator, name)]+ args, limit=limit, context=context)
```

```
        else:
            ids = self.search(cr, user, args, limit=limit, context=context)
        return self.name_get(cr, user, ids, context=context)
```

#关联客户时将客户名称自定义显示为“名称 手机”

```
    def name_get(self, cr, uid, ids, context=None):
        if not ids:
            return []
        if isinstance(ids, (int, long)):
                    ids = [ids]
        reads = self.read(cr, uid, ids, ['name', 'mobile'], context=context)
        res = []
        for record in reads:
            name = record['name']
            if record['mobile']:
                name = name + ' ' + record['mobile']
            res.append((record['id'], name))
        return res
```

2.4.10 缺省值存取方法

说明：default_get, default_set

def name_get(self, cr, uid, ids, context={})

def name_search(self, cr, uid, name=, args=[], operator=’ ilike’ , context={})

2.4.11 create 方法

说明：在数据表中插入一条记录（或曰新建一个对象的 resource）。

格式：def create(self, cr, uid, vals, context={})

参数说明：

vals: 待新建记录的字段值，是一个字典，形如: {'name_of_the_field':value, ...}

context (optional): Odoo 几乎所有的方法都带有参数 context，context 是一个字典，存放一些上下文值，例如当前用户的信息，包括语言、角色等。context 可以塞入任何值，在 action 定义中，有一个 context 属性，在界面定义时，可以在该属性中放入任何值，context 的最初值通常来自该属性值。

返回值：新建记录的 id。

举例：id = pooler.get_pool(cr.dbname).get('res.partner.event').create(cr, uid,{'name': 'Email sent through mass mailing','partner_id': partner.id,'description': 'The Description for Partner Event'})

2.4.12 search 方法

查询符合条件的记录。

格式：def search(self, cr, uid, args, offset=0, limit=2000)

参数说明：

args: 包含检索条件的 tuples 列表，格式为: [('name_of_the_field', 'operator', value), ...]。可用的 operators 有：=, >, <, <=, >= in like, ilike child_of

更详细说明，参考《Odoo 应用和开发基础》中的“域条件”有关章节。

offset (optional): 偏移记录数，表示不返回检索结果的前 offset 条

limit (optional): 返回结果的最大记录数

返回值：符合条件的记录的 id list。

2.4.13 read 方法

返回记录的指定字段值列表。

格式：def read(self, cr, uid, ids, fields=None, context={})

参数说明：

ids: 待读取的记录的 id 列表，形如[1,3,5,...]

fields (optionnal): 待读取的字段值，不指定的话，读取所有字段。

context (optional): 参见 create 方法。

返回值：返回读取结果的字典列表，形如 [{'name_of_the_field': value, ...}, ...]

2.4.14 browse 方法

浏览对象及其关联对象。从数据库中读取指定的记录，并生成对象返回。和 read 等方法不同，本方法不是返回简单的记录，而是返回对象。返回的对象可以直接使用"."存取对象的字段和方法，形如"object.name_of_the_field"，关联字段(many2one 等)，也可以通过关联字段直接访问“相邻”对象。例如：

```
addr_obj = self.pool.get('res.partner.address').browse(cr, uid, contact_id)
nom = addr_obj.name
compte = addr_obj.partner_id.bank
```

这段代码先从对象池中取得对象 res.partner.address，调用它的方法 browse，取得 id=contact_id 的对象，然后直接用"."取得"name"字段以及关联对象 patner 的银行(addr_obj.partner_id.bank)。

格式：def browse(self, cr, uid, select, offset=0, limit=2000)

参数说明：

select: 待返回的对象 id，可以是一个 id，也可以是一个 id 列表

offset (optional): 参见 search 方法

limit (optional): 参见 search 方法

返回值：返回对象或对象列表。

注意：本方法只能在 Server 上使用，由于效率等原因，不支持 rpc 等远程调用。

2.4.15 write 方法

说明：保存一个或几个记录的一个或几个字段。

格式：def write(self, cr, uid, ids, vals, context={})

参数说明：

ids: 待修改的记录的 id 列表

vals: 待保存的字段新值，是一个字典，形如: {'name_of_the_field': value, ...}

context (optional): 参见 create 方法

返回值：如果没有异常，返回 True，否则抛出异常。

举例：self.pool.get('sale.order').write(cr, uid, ids, {'state':'cancel'})

2.4.16 unlink 方法

说明：删除一个或几个记录。

格式：def unlink(self, cr, uid, ids)

参数说明：

ids: 待删除的记录的 id 列表。

返回值：如果没有异常，返回 True，否则抛出异常

2.4.17 default_get 方法

说明：复位一个或多个字段的缺省值。

格式: def default_get(self, cr, uid, fields, form=None, reference=None)

参数说明:

fields: 希望复位缺省值的字段列表。

form (optional): 目前似乎未用(5.06 版)。

reference (optional): 目前似乎未用(5.06 版)。

返回值: 字段缺省值，是一个字典，形如： {'field_name': value, ... }。

举例：self.pool.get('hr.analytic.timesheet').default_get(cr, uid, ['product_id','product_uom_id'])

2.4.18 default_set 方法

说明：重置字段的缺省值。

格式: def default_set(self, cr, uid, field, value, for_user=False)

参数说明:

field: 待修改缺省值的字段。

value: 新的缺省值。

for_user (optional): 修改是否只对当前用户有效，还是对所有用户有效，缺省值是对所有用户有效。

返回值: True

2.4.19 错误、警告、提示

检查业务逻辑中的错误，终止代码执行，显示错误或警告信息：

```
raise osv.except_osv(_('Error!'), _('Error Message.'))
```

示例代码：

```
#删除当前销售单，需要验证销售单的状态
def unlink(self, cr, uid, ids, context=None):
        for rec in self.browse(cr, uid, ids, context=context):
            if rec.state not in ['draft']:
                raise osv.except_osv(_(u'警告!'),_(u'您不能删除以下状态的销售单 %s .')%(rec.state))
            if rec.create_uid.id != uid:
                raise osv.except_osv(_(u'警告!'),_(u'您不能删除他人创建的单据.'))
        return super(sale, self).unlink(cr, uid, ids, context)
```

字段的 onchange 事件中返回值，同时返回提示信息：

```
warning = {
    'title': _('Warning!'),
    'message' : _('Warning Message.')
    }
return {'warning': warning, 'value': value}
```

示例代码：

```
    def onchange_pricelist_id(self, cr, uid, ids, pricelist_id, order_lines, context=None):
        context = context or {}
        if not pricelist_id:
            return {}
        value = {
            'currency_id': self.pool.get('product.pricelist').browse(cr, uid, pricelist_id, context=context).currency_id.id
        }
        if not order_lines:
            return {'value': value}
        warning = {
            'title': _('Pricelist Warning!'),
            'message' : _('If you change the pricelist of this order (and eventually the currency), prices of existing order lines will not be updated.')
        }
        return {'warning': warning, 'value': value}
```

视图中 button 按钮点击时显示确认信息：

<button name="cancel_voucher" string="Cancel Voucher" type="object" states="posted" confirm="Are you sure you want to unreconcile this record?"/>

2.4.1 日期时间方法

日期格式化字符串：DATE_FORMAT = "%Y-%m-%d"

日期时间格式字符串：DATETIME_FORMAT = "%Y-%m-%d %H:%M:%S"

日期时间格式字符串（包含毫秒）：DATETIME_FORMAT = "%Y-%m-%d %H:%M:%S.%f"

Odoo 对象中字段赋值为当前日期（字符串）：fields.date.context_today，fields.date.context_today(self, cr, uid, context=context)，fields.date.today()

Odoo 对象中字段赋值为当前时间（字符串）：fields.datetime.now()，fields.datetime.context_timestamp(cr, uid, datetime.now(), context=context)

Odoo 官方建议 date/datetime 的默认值的写法是：fields.date.context_today，fields.datetime.now()

字符串转换为日期时间：datetime.datetime.strptime(sale.date, DATE_FORMAT)

日期时间转换为字符串：datetime.datetime.strftime(datetime.date.today(), DATE_FORMAT)

python 中获取当前日期：datetime.date.today()

python 中获取当前时间：datetime.datetime.now()

应用示例代码：

#自动获取日期对应的月份并保存

```
    def _get_month(self, cr, uid, ids, field_name, arg, context=None):
        res = {}
        if context is None:
            context = {}
        DATETIME_FORMAT = "%Y-%m-%d"
        for sale in self.browse(cr, uid, ids, context=context):
            saledate = datetime.datetime.strptime(sale.date, DATETIME_FORMAT)
            res[sale.id] = saledate.strftime('%Y') + '-' + saledate.strftime('%m')
        return res

    _columns={
        'name':fields.char(u'单号', size=64, select=True, required=True, readonly=True),
        'date':fields.date(u'日期', select=True, required=True, readonly=True),
        'month':fields.function(_get_month, method=True, type='char', size=10, string = u'月份', store=True, invisible=True),
    }
    _defaults={
        'name': lambda obj, cr, uid, context: '/',
```

```
        'date':fields.date.context_today,
        #'employee_id':_employee_get,
        'state':'draft'
    }
```

#自动计算到期日期，按开卡日期加 年数*365 天

```
    def _get_due_date(self, cr, uid, ids, field_name, arg, context=None):
        res = {}
        if context is None:
            context = {}
        DATE_FORMAT = "%Y-%m-%d"
        for rec in self.browse(cr, uid, ids, context=context):
            category = rec.category
            if category:
                remaining_times=category.times_limit
                if rec.active_date:
                    res[rec.id]=(datetime.datetime.strptime(rec.active_date, DATE_FORMAT) +
datetime.timedelta(days=category.age_limit*365)).strftime(DATE_FORMAT)
                else:
                    res[rec.id]=(datetime.date.today()+
datetime.timedelta(days=category.age_limit*365)).strftime(DATE_FORMAT)
        return res

    _columns={
        "name":fields.char("卡号",size=64, required=True, readonly=True, states={'0':[('readonly',False)]}),
        "category":fields.many2one("dispatch.service_card_category","服务卡类型", required=True, readonly=True,
states={'0':[('readonly',False)]}),
        'age_limit':fields.related('category', 'age_limit', string=u'年限', type='float', readonly=True, store=True),
        "times_limit":fields.integer(u"初始次数", readonly=True),
        "customer":fields.many2one("dispatch.customer","客户",required=True, select=True, readonly=True,
states={'0':[('readonly',False)]}),
        "remaining_times":fields.integer("剩余次数",required=True, readonly=True, states={'0':[('readonly',False)]}),
        "active_date":fields.date("开卡日期",required=True, readonly=True, states={'0':[('readonly',False)]}),
        'due_date':fields.function(_get_due_date, method=True, type='date', string = u'到期日期', store=True),
        'state': fields.selection([('0', u'未开卡'),('1', u'已开卡'),('2', u'已用完'),('3', u'已过期'),('4', u'已锁定')], u'状态
',required=True, readonly=True),
        'lock_note': fields.char(u'锁定原因', size=200, invisible=False, readonly=True, states={'1':[('readonly',False)],
```

```
'4':[('readonly',False)]}),
    }

    # TODO: can be improved using resource calendar method
```

#计算日期间隔对应的天数

```
    def _get_number_of_days(self, date_from, date_to):
        """Returns a float equals to the timedelta between two dates given as string."""

        DATETIME_FORMAT = "%Y-%m-%d %H:%M:%S"
        from_dt = datetime.datetime.strptime(date_from, DATETIME_FORMAT)
        to_dt = datetime.datetime.strptime(date_to, DATETIME_FORMAT)
        timedelta = to_dt - from_dt
        diff_day = timedelta.days + float(timedelta.seconds) / 86400
        return diff_day
```

#对象字段

```
    _columns = {
            'date_from': fields.datetime(u'起始日期',required=True, readonly=True, states={'draft':[('readonly',False)]},
select=True),
            'date_to': fields.datetime(u'结束日期', readonly=True, states={'draft':[('readonly',False)]}),
            'days': fields.float(u'天数', digits=(8, 2), readonly=True, states={'draft':[('readonly',False)]}),
          }
```

#更改起始日期，自动计算请假天数

```
    def onchange_date_from(self, cr, uid, ids, date_to, date_from):
        """
        If there are no date set for date_to, automatically set one 8 hours later than
        the date_from.
        Also update the number_of_days.
        """
        # date_to has to be greater than date_from
        if (date_from and date_to) and (date_from > date_to):
            raise osv.except_osv(_(u'警告!'),_(u'开始日期必须小于结束日期.'))

        result = {'value': {}}

        # No date_to set so far: automatically compute one 8 hours later
        if date_from and not date_to:
```

```
            date_to_with_delta = datetime.datetime.strptime(date_from, tools.DEFAULT_SERVER_DATETIME_FORMAT)
+ datetime.timedelta(hours=8)
            result['value']['date_to'] = str(date_to_with_delta)

        # Compute and update the number of days
        if (date_to and date_from) and (date_from <= date_to):
            diff_day = self._get_number_of_days(date_from, date_to)
            result['value']['days'] = round(math.floor(diff_day))+1
        else:
            result['value']['days'] = 0

        return result
```

#更改结束日期，自动计算请假天数

```
    def onchange_date_to(self, cr, uid, ids, date_to, date_from):
        """
        Update the number_of_days.
        """
        # date_to has to be greater than date_from
        if (date_from and date_to) and (date_from > date_to):
            raise osv.except_osv(_(u'警告!'),_(u'开始日期必须小于结束日期.'))
        result = {'value': {}}

        # Compute and update the number of days
        if (date_to and date_from) and (date_from <= date_to):
            diff_day = self._get_number_of_days(date_from, date_to)
            result['value']['days'] = round(math.floor(diff_day))+1
        else:
            result['value']['days'] = 0

        return result
```

2.4.2 设置日期及时间格式并在后台代码中动态获取

页面设置可至 **设置->语言->简体中文->编辑** 进行日期格式及时间格式的修改

名称	Chinese (CN) / 简体中文	地区代码	zh_CN	ISO 国家代码	zh_CN
分割符格式	[]	日期格式	%Y-%m-%d	时间格式	%I:%M:%S
方向	从左到右	十进位分割符	.	千位分隔符	,
有效	☑	可翻译	☑		

设置完后如果需要在后台代码中动态获取以上配置可参考如下代码

```
# 获取配置表
pool_lang = self.pool.get('res.lang')
# 从context中获取当前语言并作为搜索条件获取语言配置对象
lang_ids = pool_lang.search(cr, uid, [('code', '=', context['lang'])])[0]
lang_obj = pool_lang.browse(cr,uid,lang_ids)
# 从语言配置对象中获取日期格式及时间格式
datetime_format = lang_obj.date_format + " " + lang_obj.time_format
```

2.4.3 context 的应用

在 Action 中定义，context 用于传递搜索条件和分组条件，在搜索视图中默认显示：

<field name="context">{'search_default_group_ is_company ': 1, 'search_default_ customer ': 1}</field>

定义了窗口打开时默认的分组条件 is_company 和过滤条件 customer ，如图所示：

在 Action 中定义，context 用于传递创建对象的默认值：

<act_window

```
id="act_crm_opportunity_crm_phonecall_new"
name="Phone calls"
groups="base.group_sale_salesman"
res_model="crm.phonecall"
view_mode="tree,calendar,form"
context="{'default_duration': 1.0 ,'default_opportunity_id': active_id}"
view_type="form"/>
```

在搜索视图中定义，context 用于传递在视图中使用的变量：

```
<filter string="Show Countries" context="{'invisible_country': False}" help="Show Countries"/>
```

定义了在搜索视图中选择“显示国家”时，设置变量 invisible_country 的值为 False，tree 视图中引用变量，设置字段“国家”的显示或隐藏，

tree 视图中的代码如下所示：

```
<field name="country_id" invisible="context.get('invisible_country', True)"/>
```

在 Action 中定义，context 用于传递在后台搜索方法中使用的变量：

```
context="{'related_product':1}"
```

在后台代码中传递，context 用于方法调用过程中变量的传递：

```
    #写入，可用于校验写入和更改数据的合法性
    def write(self, cr, uid, ids, vals, context=None):
        if context is None:
            context = {}
        if not context.get('set_to_draft', False): #非设置为草稿状态的操作
            for rec in self.browse(cr, uid, ids, context=context):
                if rec.state == 'draft' and rec.create_uid.id != uid:
                    raise osv.except_osv(_(u'警告!'),_(u'您不能修改他人创建的单据.'))
        return super(dispatch_sale, self).write(cr, uid, ids, vals, context=context)
```

#设置为草稿状态

```
    def set_to_draft(self, cr, uid, ids, context=None):
        ctx = context.copy()
        ctx['set_to_draft'] = True #用于在 write 方法中判断是否是设置为草稿状态的操作
        return self.write(cr, uid, ids, {'state': 'draft'}, ctx)
```

其中：

set_to_draft 方法中定义了变量 ctx['set_to_draft'] = True 并作为参数传递到 write 中，

write 方法中判断变量的值 if not context.get('set_to_draft', False) 执行响应的业务逻辑。

2.4.4 关系字段类型

OpenObject 默认保留了几个名称，他们由系统自动创建，创建相同名称都会被忽略

id	系统自动创建
name	默认情况用于显示这个列表的值，你可以使用 _rec_name 构建自己想要显示的列表

练习 2 - 定义一个模型

定义一个新的会议数据模型。会议有一个名字，或“标题”，和描述。会议名称是必需的

第3章 菜单操作

3.1 Actions

触发三个方面事件:

1. 通过点击菜单，链接到一个特定的动作
2. 点击视图中的按钮，连接到指定动作
3. 作为一个对象的连接上下文操作

菜单关联到窗口动作上，根据窗口视图顺序来显示特定的窗体

下面的实例是创建一个菜单，并把他关联到列表和表单中 ，注意菜单上有几个属性。

```
<record model="ir.actions.act_window" id="action_list_ideas">
    <field name="name">Ideas</field>
    <field name="res_model">idea.idea</field>
    <field name="view_mode">tree,form</field>
</record>
```

record 属性

- model 固定属性 ir.actions.act_window
- id 唯一
- name 动作名称
- res_model 对象
- view_mode 显示形式可以对调来优先显示不同视图

```
<menuitem id="menu_ideas" parent="menu_root" name="Ideas" sequence="10" action="action_list_ideas"/>
```

menuitem 属性

- id 不能重名
- parent 为上级菜单，如果菜单不在同一个 XML 要加上文件夹名称
- name 界面上显示的文字
- sequence 菜单的排序
- action 触发的事件

Note

Action 通常是执行 XML 文件中对应的菜单，因为 action_id 存在于数据库中，你可以创建一个菜单指定到 Action 上

练习 3 - 定义菜单

定义一个新的菜单项来访问会议模块，不安装

显示列表的所有会议

创建/修改会议

第4章 基本知识

练习 1 - 通过 Web 界面视图编辑器自定义视图

1. 使用会议模型创建的树视图，显示会议的名称及其说明。
2. 通过时间先后顺序来排序。
3. 声明一个继承视图（浏览继承）。

视图有三种类型的标签：

- <record> 标签有属性 model= “ir.ui.view” ，它包含本身的视图定义
- <record> 标签有属性 model= “ir.actions.act_window” ，它将动作和这些视图链接起来。
- <menuitem> 标签，在菜单中创建记录，将动作链接起来。

4.1 声明视图类型

界面视图使用 XML 来设计

```
<record model="ir.ui.view" id="view_id">
    <field name="name">view.name</field>
    <field name="model">object_name</field>
    <field name="type">form</field>      <!-- tree,form,calendar,search,graph,gantt -->
    <field name="priority" eval="16"/>
    <field name="arch" type="xml">
     <!-- view content: <form>, <tree>, <graph>, ... -->
    </field>
</record>
```

Form view 属性
➢ model 固定属性 ir.ui.view ➢ id 唯一 ➢ Name 视图名称 ➢ model 对象 ➢ type 有多种类型 tree,form,calendar,search,graph,gantt ➢ priority 排序

请注意视图本身为 XML。因此记录必须声明 XML 类型，以解析字段的值，元素和界面。

4.2 树型视图

树视图，也称为列表视图，以表格形式显示记录。他们是 XML 元素定义的<tree>。

```
<tree string="Idea list" colors="grey:state=='cancel';blue:state in
('waiting_date','manual');red:state in ('invoice_except','shipping_except')"
fonts="bold:message_unread==True" delete="false" create="false" write="false" read="false"
copy="false" >
    <field name="name"/>
    <field name="inventor_id"/>
</tree>
```

颜色：colors="grey:state=='cancel';blue:state in ('waiting_date','manual');red:state in ('invoice_except','shipping_except')"

属性：delete="false" create="false" write="false" read="false" copy="false"

粗体：fonts="bold:message_unread==True"

4.3 表单视图

表单界面都是通过 XML 元素进行渲染，允许你创建/编辑表单。

例如：以下表单视图显示了如何使用分离器，群组，页卡等空间结构的说明文件，把有相同功能的字段放在一个组中。

Group 组的几个属性	
colspan	说明该控件占用父容器多少列 Odoo form 为顶级容器，约定为 4 列
rowspan	行数
expand	
col	用于容器控件，如 group,它表示这个容器内部分几列
string	组的名称
备注	在 Odoo 中，控件一般都有 label，每个控件(设计元素）默认是占据两列 ,在一个 group 中，控件是从左到右，用完所有总列数后，然后从上往下排列的

示例

①<group col="6" colspan="4">
②<group col="2" colspan="4">
<separator colspan="4" string="Product Description"/>
<field name="name" select="1"/>
<field groups="base.group_extended" name="variants" select="2"/>
</group>
③<group col="2" colspan="1">
<separator colspan="2" string="Codes"/>
<field name="default_code" select="1"/>
<field groups="base.group_extended" name="ean13" select="2"/>
</group>
④<group col="2" colspan="1">
<separator colspan="2" string="Product Type"/>
<field name="sale_ok" select="2"/>
<field name="purchase_ok" select="2"/>
<field groups="base.group_extended" name="rental" select="2"/>
</group>
</group>

在这个例子中，①的 colspan="4"表示窗体占据整行，col="6' 表示① 这个组控件内部分为 6 列

②号组中的 col="2"表示②这个组控件内部分 2 列，colspan="4"表示占用①组控件的 4 列

③号组中的 col="2"表示③这个组控件内部分 2 列，colspan="1"表示占用①组控件的 1 列

④号组中的 col="2"表示④这个组控件内部分 2 列，colspan="1"表示占用①组控件的 1 列

```
<form string="Idea form">
    <group colspan="2" col="2">
        <separator string="General stuff" colspan="2"/>
        <field name="name"/>
        <field name="inventor_id"/>
    </group>
    <group colspan="2" col="2">
        <separator string="Dates" colspan="2"/>
        <field name="active"/>
        <field name="invent_date" readonly="1"/>
    </group>

    <notebook colspan="4">
```

```
            <page string="Description">
                <field name="description" nolabel="1"/>
            </page>
        </notebook>

        <field name="state"/>
    </form>
```

Odoo 7.0 版本中可以写 HTML 标签

```
    <form string="Idea Form v7"   version="7.0">
        <header>
    <button string="Confirm" type="object" name="action_confirm" states="draft" class="oe_highlight" />
    <button string="Mark as done" type="object" name="action_done" states="confirmed" class="oe_highlight"/>
    <button string="Reset to draft" type="object" name="action_draft" states="confirmed,done" />
            <field name="state" widget="statusbar"/>
        </header>
        <sheet>
            <div class="oe_title">
                <label for="name" class="oe_edit_only" string="Idea Name" />
                <h1><field name="name" /></h1>
            </div>
            <separator string="General" colspan="2" />
            <group colspan="2" col="2">
                <field name="description" placeholder="Idea description..." />
            </group>
        </sheet>
    </form>
```

练习 2 - 使用 XML 定制 Tree 与 Form

创建会议对象树和表单视图。数据显示格式:

- 在树上显示会议的名称
- 在表单视图,编辑名称和描述

练习 3 - 页卡

- 把描述信息放在一个页卡中，方便扩展其它额外的信息

4.4 对象之间的关系

练习 1 - 创建一个完整类
创建类字段如下： 参加者,会议名称(需要),开始日期,持续时间(天)和一个座位的数量。 要求建一个菜单、窗口事件、一个列表、表单视图

4.5 关系字段

- 关系字段值之间的引用对象。
- 常见的属性关系
- 域:可选参数的搜索限制
- **many2one(obj, ondelete=’ set null’ ,...)**：简单的关系转向另一个对象(使用一个外键)

obj: _name 对象是必选，使用有颜色来区分

ondelete: 详细见下图 【ondelete 属性】

many2one 属性
直接填入已经存在的数据的 id 或者填入 False
例子：

'instructor_id': fields.many2one('openacademy.instructor','Instructor')

one2many(obj, field_id, ...)：一个值对应多个值,他的另一个关联是 many2one 可以查看 one2many 全部信息

obj: 对象必填

- **field_id: 字段名对象表中 many2one 字段,对应外键(必需)**

one2many 属性
(0, 0,{ values })根据 values 里面的信息新建一个记录
(1,ID,{values}) 更新 id=ID 的记录（对 id=ID 的执行 write 写入 values 里面的数据）
(2,ID) 删除 id=ID 的数据（调用 unlink 方法，删除数据以及整个主从数据链接关系）
例子：
[(0,0,{'field_name':field_value_record1,...}),(0,0,{'field_name':field_value_record})]
'training_ids': fields.one2many（ 'openacademy.training' ,instructor_id' ,'Trainings')

many2many(obj, rel, field1, field2, ...)： 双向多个关系对象。这是最一般类型的关系:一个记录可能与任何数量的记录在另一边,反之亦然。

- obj: _name 必填关联的对象
- rel: 新增的关联表
- field1: 当前对象的 ID
- field2: obj 对象 ID

注意参数 rel,field1 和 field2 是可选的。当字段未设计时,ORM 自动生成适合它们的值

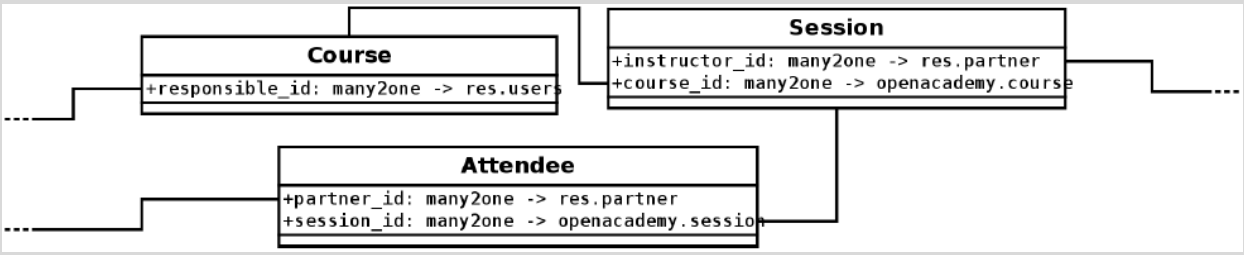
many2many 属性
(0,0,{values}) 根据 values 里面的信息新建一个记录。
(1,ID,{values})更新 id=ID 的记录（写入 values 里面的数据）
(2,ID) 删除 id=ID 的数据（调用 unlink 方法，删除数据以及整个主从数据链接关系）
(3,ID) 切断主从数据的链接关系但是不删除这个数据

(4,ID) 为 id=ID 的数据添加主从链接关系。
(5) 删除所有的从数据的链接关系就是向所有的从数据调用(3,ID)
(6,0,[IDs]) 用 IDs 里面的记录替换原来的记录（就是先执行(5)再执行循环 IDs 执行（4,ID））
例子：
例子[(6, 0, [8, 5, 6, 4])] 设置 many2many to ids [8, 5, 6, 4]
'participant_ids' : fields.many2many('obj','news_obj_rel', 'field1', 'field2','Participants')

ondelete 属性
no atction：相互不影响
cascade:主键被删除，外键对应的记录也删除。直接删除外键的记录，不影响主键。
restrict: 如果存在外键，主键删除的时候报错。
set null：主键被删除，外键变为空值。
set default:主键被删除，外键变为默认值。
例子：
ondelete=’ cascade’
postgresSQL 本身有的特性 http://www.postgresql.org/docs/8.2/static/ddl-constraints.html

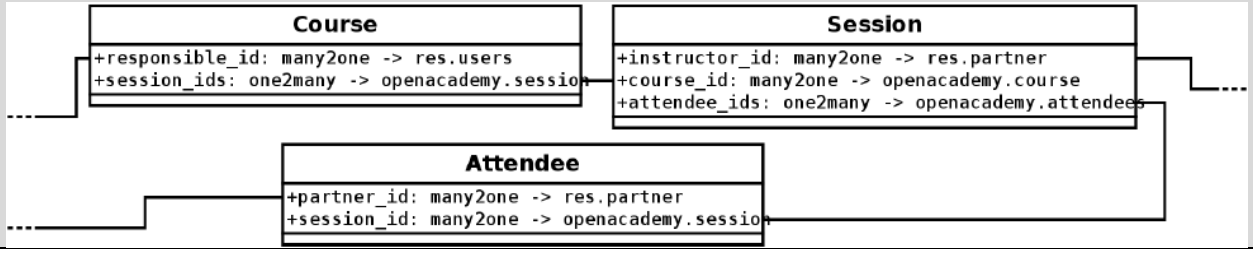
练习 2 - 关系 many2one

使用 many2one 关系字段,修改类会议、会话和参加者以反映其与其他对象的关系,定义如下



练习 3 - 关系 one2many 视图

使用的的逆关系领域 one2many，修改类会议，会话和与会者，以反映他们的关系，与其他对象，其定义如下。



练习 4 - 修改视图

在 Tree 中显示会议对象名称各负责该会议的老师

在表单顶部把把字段加上描述一起放到第二个页卡中

4.6 字段 CSS 对齐方式

首先在模块文件__openerp__.py中添加一个自定义css文件

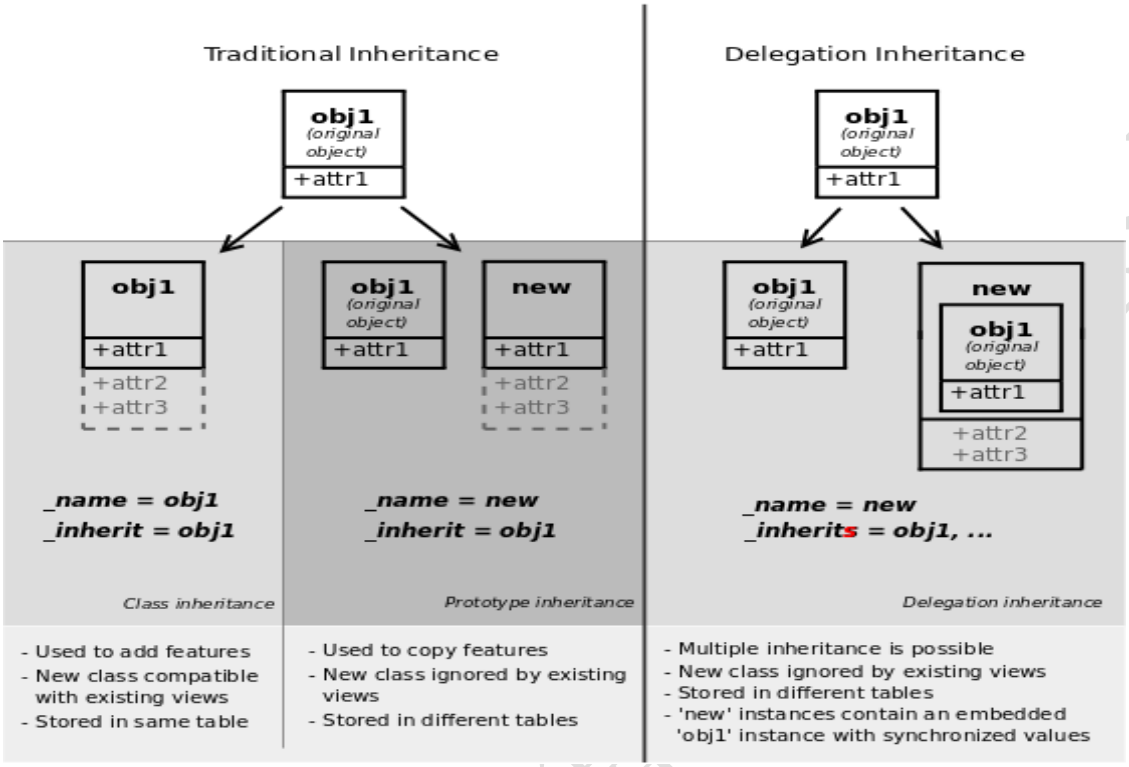
```
'css': [
    'static/src/css/show.css'
],
```

之后创建并添加如下样式到show.css文件中

```
.openerp .oe_list_content td.oe_number {
    text-align: left !important;
}
```

第5章 继承

5.1 继承机制



预定义的 osv.Model 业务对象的属性

_inherit	_name 上级业务对象（继承原型）
_inherits	对于多个/实例继承机制:字典映射业务对象所在的名称的名称对应的外键字段名称使用

5.2 查看继承

如果要修改一个已知的模块，正解的做法是：通过继承视图来修改添加元素使用 XPath 表达式来定位

属性	attributes：更改属性的标签 inside（默认）：你的价值观将被附加在标签内 before：前的内容标签 after：后添加内容标签 replace：更换标签内容。
修改属性：修改属性能够实现的功能，不要使用 replace	

XPath 参考中可以找到 www.w3.org/TR/xpath

支持的定位方法：

<notebook position="inside">

<xpath expr="//page[@name='page_history']" position="inside">

<field name="mobile" position="after">

<filter name="consumable" position="after">

<group name="bank" position="after">

<xpath expr="//field[@name='standard_price']" position="replace">

<xpath expr="//button[@name='open_ui']" position="replace">

<xpath expr="//div[@class='oe_employee_details']/h4/a" position="after">

<xpath expr="/form/sheet/notebook/page/field[@name='line_ids']/tree/field[@name='analytic_account_id']" position="replace">

<xpath expr="/form/sheet/notebook/page/field[@name='line_ids']/form/group/field[@name='analytic_account_id']" position="replace">

5.3 实例

```
<record model="ir.ui.view" id="view_partner_form">
    <field name="name">res.partner.form.inherit</field>
    <field name="model">res.partner</field>
    <field name="inherit_id" ref="base.view_partner_form"/>
    <field name="arch" type="xml">
        <notebook position="inside">
            <page string="Relations">
                <field name="relation_ids" colspan="4" nolabel="1"/>
            </page>
        </notebook>
    </field>
</record>
```

注意：所有的操作只能在下面包含内

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

系统有四种属性来使用

- **inside**（默认）：你的价值观将被附加在标签内
- **after**：后添加内容标签
- **before**：前的内容标签
- **replace**：更换标签内容。

5.3.1 更换内容

```
<record model="ir.ui.view" id="view_partner_form1">
    <field name="name">res.partner.form.inherit1</field>
    <field name="model">res.partner</field>
    <field name="inherit_id" ref="base.view_partner_form"/>
    <field name="arch" type="xml">
        <page string="Extra Info" position="replace">
            <field name="relation_ids" colspan="4" nolabel="1"/>
        </page>
    </field>
</record>
```

5.3.2 删除内容

```
<record model="ir.ui.view" id="view_partner_form2">
    <field name="name">res.partner.form.inherit2</field>
    <field name="model">res.partner</field>
    <field name="inherit_id" ref="base.view_partner_form"/>
    <field name="arch" type="xml">
        <field name="lang" position="replace"/>
    </field>
</record>
```

5.3.3 插入内容

```
<record model="ir.ui.view" id="view_partner_form3">
    <field name="name">res.partner.form.inherit3</field>
    <field name="model">res.partner</field>
    <field name="inherit_id" ref="base.view_partner_form"/>
    <field name="arch" type="xml">
        <field name="lang" position="before">
            <field name="relation_ids"/>
        </field>
    </field>
</record>
<record model="ir.ui.view" id="view_partner_form4">
    <field name="name">res.partner.form.inherit4</field>
```

```
        <field name="model">res.partner</field>
        <field name="inherit_id" ref="base.view_partner_form"/>
        <field name="arch" type="xml">
            <field name="lang" position="after">
                <field name="relation_ids"/>
            </field>
        </field>
    </record>
```

5.3.4 多变化

```
    <record model="ir.ui.view" id="view_partner_form5">
        <field name="name">res.partner.form.inherit5</field>
        <field name="model">res.partner</field>
        <field name="inherit_id" ref="base.view_partner_form"/>
        <field name="arch" type="xml">
            <data>
                <field name="lang" position="replace"/>
                <field name="website" position="after">
                    <field name="lang"/>
                </field>
            </data>
        </field>
    </record>
```

5.3.5 XPath 的节点

```
    <record model="ir.ui.view" id="view_partner_form6">
        <field name="name">res.partner.form.inherit6</field>
        <field name="model">res.partner</field>
        <field name="inherit_id" ref="base.view_partner_form"/>
        <field name="arch" type="xml">
            <data>
                <xpath expr="//field[@name='address']/form/field[@name='email']"
                    position="after">
                    <field name="age"/>
                </xpath>
                <xpath
```

```
                    expr="//field[@name='address']/tree/field[@name='email']"
                    position="after">
                    <field name="age"/>
                </xpath>
            </data>
        </field>
    </record>
    定位
    <xpath expr="/form/sheet/div[@class='oe_left']/div[@class='oe_left']/div[last()]" position="inside">
        <div>
            <field name="name"/>
        </div>
    </xpath>
```

练习 1 - 添加一个继承机制
使用类继承,创建一个“伙伴”类修改现有的“伙伴”类,添加一个“是否老师“布尔字段，在列表中也要加入这个 字段。使用视图继承——例如,修改现有的合作伙伴视图以显示新的字段

5.4 域 Domain

转义符：

注意：在 xml 文件中使用时 Domain 不能直接使用<, >等符号作为 domain 的 operator。 这个不是 Odoo 的问题，而是 xml 标准的问题

> 开始标记 >

< 结束标记 <

" 引号 "

' 撇号 '

& "&"符 &

>= >=

<= <=

!= ><

5.4.1.1 Domin 条件表达式规则

domain 中的单个条件是一个三个元素组成的元组。

domain=[('hr_expense_ok','=',True)] 最简单的格式：[('字段', '操作符', 值)]

- 第一个是对象的一个 column，也就是字段名；

- 第二个是比较运算符``=, !=, >, >=, <, <=, like, ilike, in, not in, child_of, parent_left, parent_right``；
- 第三个就是用来比较的值了。

多个条件用“|”（or），“&”（and），“!”（no）逻辑运算符链接。

逻辑运算符作为前缀放置于条件前面。“|”与“&”必须两个条件链接，“!”对一个条件取反。默认逻辑运算符为“&”。

例如,下面的域选择所有产品类型的服务,有一个 unit_price 大于 1000。

[('product_type', '=', 'service'), ('unit_price', '>', 1000)]

下面的例子编码条件:“类型的服务或单价在 1000 年和 2000 之间”。

['|', ('product_type', '=', 'service'), '&', ('unit_price', '>=', '1000'), ('unit_price', '<', '2000')]

举个例子：

['|','|',('group_ids','in',[g.id for g in user.groups_id]), ('user_id', '=', user.id), '&', ('user_id', '=', False), ('group_ids','=',False), '|','|', ('company_id','=',False), ('company_id','child_of',[user.company_id.id]),('company_id.child_ids','child_of',[user.company_id.id])]

这个例子的意思是：

['|',

'|',('group_ids','in',[g.id for g in user.groups_id]), ('user_id', '=', user.id),

'&', ('user_id', '=', False), ('group_ids','=',False),

'|',

'|', ('company_id','=',False), ('company_id','child_of',[user.company_id.id]),

('company_id.child_ids','child_of',[user.company_id.id])]

写个容易看的方式：

(('group_ids','in',[g.id for g in user.groups_id]) or ('user_id', '=', user.id)) or (('user_id', '=', False) and ('group_ids','=',False)) or (('company_id','=',False) or ('company_id','child_of',[user.company_id.id])) or ('company_id.child_ids','child_of',[user.company_id.id])

5.4.1.2 不带逻辑运算符的简单表达式

过滤状态为草稿： [('state', '=', ' draft')]

过滤状态为草稿、取消：

[('state', 'in', ('draft', 'cancel'))] 或者 [('state', 'in', ['draft', 'cancel'])]

过滤金额大于 5000： [('price_unit', '>', 5000)]

5.4.1.3 带逻辑运算符的简单表达式

假设 a , b 分别是不带逻辑运算符的简单表达式

a = ('state','><','review')

b = (amount_total, '<=', 2000)

a and b: [a, b] 或 ['&amp' , a , b]

a or b: ['|', a,b]

Note:

销售经理登陆销售订单时待审批的销售订单（状态是待批，并且总金额在大于 2000）菜单过滤条件：

<field name="domain">[('state','=','wait_prove'),('amount_total', '> ', 2000)]</field>

5.4.1.4 带逻辑运算符的表达式

a and b and c : [a, b, c] 或则 ['&', '&', a, b, c]

a or b or c: ['|', '|', a , b , c]

a and b or c: ['|', '&', a, b, c]

a and (b or c): ['&', a, '|', b, c]

Note:

经理待审批的请假单（状态是待批，并且请假天数大于一天，并且是本部门的职员请假单，并且还不包含自己的请假单）菜单过滤条件

<field name="domain"> [('state', '=', 'wait_prove'), ('tians', '>', 1), ('shenqr.user_id','<>',uid),('shenqr.department_id','=',department_id),('shenqr.user_id.groups_id', '=', 59)] </field>

5.4.1.5 带逻辑运算符复杂的表达式

同上，假设 a , b, c, e, f, g 分别是不带逻辑运算符的简单表达式

(a or b and c) or (d and e) :

['|', '&','|', a, b, c, '&', d, e, 3)]

Note:

总经理待审批的请假单（所有部门副经理或经理状态为待批的请假单，或 3 天以上部门经理批准过的请假单）菜单过滤条件：

<field name="domain">['|', '&','|', ('shenqr.user_id.groups_id', '=', 60), ('shenqr.user_id.groups_id', '=', 61), ('state', '=', 'wait_prove'), '&', ('state','=','depmanager_proved'),('tians', '>', 3)]</field>

(a or b and c) or (d and e) or (f and g)

['|','|', '&','|', a, b, c, '&', d,e,'&', f,g]

Note:

总经理全部的请假单（所有部门副经理或经理状态为待批的请假单，或 3 天以上部门经理批准过的请假单，或 3 天以上状态为同意或驳回的请假单据）菜单过滤条件：

<field name="domain">['|','|', '&','|', ('shenqr.user_id.groups_id', '=', 60), ('shenqr.user_id.groups_id', '=', 61), ('state', '=', 'wait_prove'), '&', ('state','=','depmanager_proved'),('tians', '>', 3),'&', ('state','in',['proved','rejected']),('tians', '>', 3)]</field>

5.4.2 在 Action 中定义，domain 用于对象默认的搜索条件：

<field name="domain">[('state','not in',('draft','sent','cancel'))]</field>

定义了打开订单窗口时仅搜索不处于（'draft','sent','cancel'）三种状态的订单。

在对象（或视图）的关联字段（many2one 和 many2many 类型字段）中定义，字段值是关联表的 id，domain 用于过滤关联表的记

录

'product_id': fields.many2one('product.product', 'Product', domain=[('sale_ok', '=', True)], change_default =True),

定义了关联选择产品时，仅显示可销售的产品。

5.4.3 在搜索视图中定义，domain 用于自定义的搜索条件：

<field name="name" string="Sales Order" filter_domain="['|',('name','ilike',self),('client_order_ref','ilike',self)]"/>

定义了在搜索视图中输入订单编号时，同时按订单编号和客户关联编号过滤。

<filter string="My Sales Orders" domain="[('user_id','=',uid)]" help="My Sales Orders" icon="terp-personal" name="my_sale_orders_filter"/>

定义了在搜索视图中选择"My Sales Orders"标签时，过滤销售员是当前登录用户的订单。

5.4.4 在记录规则中定义，domain 用于定义用户对对象中记录访问的权限：

<field name="domain_force"> ['|',('company_id','=',False),('company_id','child_of',[user.company_id.id])] </field>

定义了用户仅允许查询订单中未指定分公司或订单中指定的分公司用户具有访问权限的销售订单。

['|',('section_id','=',user.default_section_id.id),('section_id','=',False)]

自己看自己的团队

练习 1 - domain
添加一个规则，表单下拉的值只能选择合作伙伴字段是老师，不是弹出提醒

练习 2 - domain
添加一个规则，表单下拉时只显示合作伙伴为教师

第6章 ORM 方法

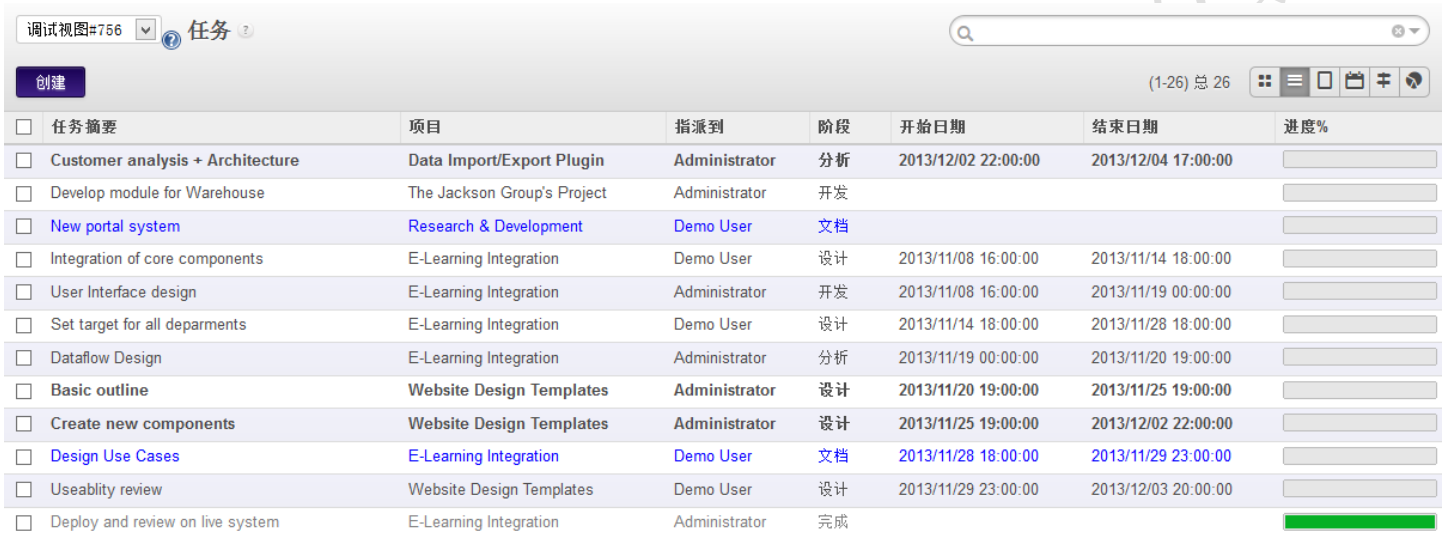
6.1 Functional 字段

上面曾经讲过 function 的用法

function(fnct, arg=None, fnct_inv=None, fnct_inv_arg=None, type='float', fnct_search=None, obj=None, store=False, multi=False, ...) :

练习 1 - Functional 字段

修改坐席以百分比形式出现，并用进度条来展示



任务摘要	项目	指派到	阶段	开始日期	结束日期	进度%
Customer analysis + Architecture	Data Import/Export Plugin	Administrator	分析	2013/12/02 22:00:00	2013/12/04 17:00:00	
Develop module for Warehouse	The Jackson Group's Project	Administrator	开发			
New portal system	Research & Development	Demo User	文档			
Integration of core components	E-Learning Integration	Demo User	设计	2013/11/08 16:00:00	2013/11/14 18:00:00	
User Interface design	E-Learning Integration	Administrator	开发	2013/11/08 16:00:00	2013/11/19 00:00:00	
Set target for all deparments	E-Learning Integration	Demo User	设计	2013/11/14 18:00:00	2013/11/28 18:00:00	
Dataflow Design	E-Learning Integration	Administrator	分析	2013/11/19 00:00:00	2013/11/20 19:00:00	
Basic outline	Website Design Templates	Administrator	设计	2013/11/20 19:00:00	2013/11/25 19:00:00	
Create new components	Website Design Templates	Administrator	设计	2013/11/25 19:00:00	2013/12/02 22:00:00	
Design Use Cases	E-Learning Integration	Demo User	文档	2013/11/28 18:00:00	2013/11/29 23:00:00	
Useablity review	Website Design Templates	Demo User	设计	2013/11/29 23:00:00	2013/12/03 20:00:00	
Deploy and review on live system	E-Learning Integration	Administrator	完成			

6.2 Onchange

```
<!-- on_change example in xml -->
<field name="amount" on_change="onchange_price(unit_price,amount)" />
<field name="unit_price" on_change="onchange_price(unit_price,amount)" />
<field name="price" />

    def onchange_price(self, cr, uid, ids, unit_price, amount, context=None): ''
    '' 修改单价时金额变化 ''''
    return {'value' :{'price' : unit_price * amount}}
```

练习 2 - Onchange 方法

修改表单视图和类以，使用百分比显示到场率，可提供座位数或出席人数对比

练习 3 - 警告

如果座位为零，选择时弹出警告

6.3 预定义 osv.Model 对象的属性

_constraints	使用 Python 进行数据验证
_sql_constraints	定义 SQL 约束

练习 4 - 添加约束

添加一个约束检查该会议名称不可以相同

练习 5 - 添加 SQL 约束

添加一个 SQL 约束检查，会议有一个唯一的名称

练习 6 - 添加一个重复的选项

既然我们增加了一个约束的会议名称的唯一性，它是不可能（名称“重复”）。重新实现自己的“复制”的方法，在新的数据名称上加上“复制”二字

练习 7 - 设置活动对象 - 默认值

定义 start_date 默认值为今天

第7章 高级视图

7.1 列表 & 树

列表包括字段元素，树型<tree>含有父元素。

属性	colors: 颜色列表映射到 Python 的条件 editable: 在顶部或底部允许编辑 toolbar: 设置为 True 工具栏显示对象层次结构的顶层（例如：菜单）
允许元素	field, group, separator, tree, button, filter, newline

```
<tree string="Idea Categories" toolbar="1" colors="blue:state==draft">
    <field name="name"/>
    <field name="state"/>
</tree>
```

练习 1 - 列表颜色:

修改树视图：持续时间少于 5 天的会议为蓝色，并持续 15 天以上的为红色.

Note:

持续时间如果为空，系统会报错

7.2 日历

使用日期字段作为日历事件。

属性	color: 字段的颜色 date_start: 包含事件开始日期/时间字段名称 date_stop: 字段名称包含事件停止日期/时间 day_length: 日历天以小时为单位的长度 (default: 8) date_delay: name of field containing event duration
允许元素	field (to define the label for each calendar event)

```
<calendar color="user_id" date_delay="planned_hours" date_start="date_start"
string="Tasks">
    <field name="name"/>
    <field name="project_id"/>
</calendar>
```

下面是该视图标签的属性:

string

该视图的标题

date_start

表示开始时间的属性，该字段是必须的。

date_stop

表示结束时间的属性，如果指定了 date_delay 属性就可以忽视该属性。

date_delay

数字字段，以时指定的时间进行记录。此属性将得到优先于 date_stop date_stop 将被忽略.

day_length

显示工作时间长度的数字值，默认为 8 小时

color

显示一个字段在日历上

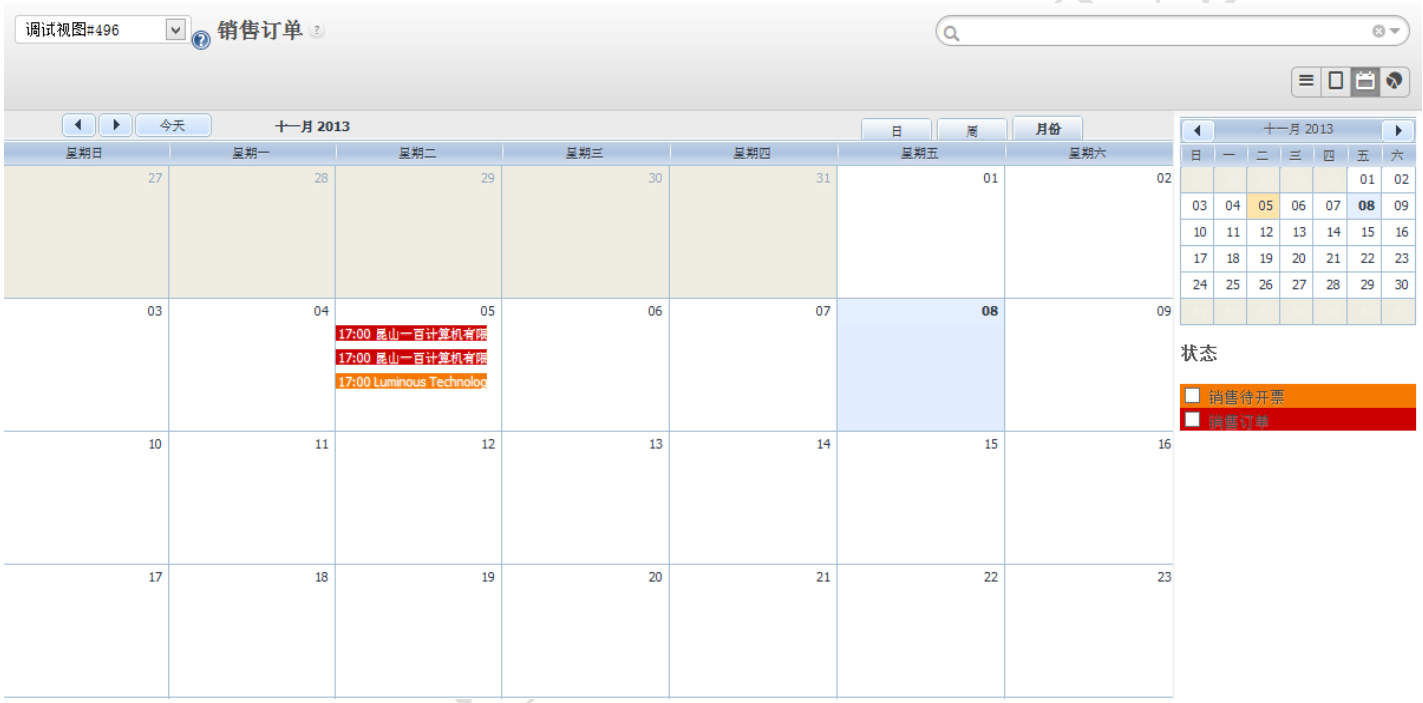
mode

默认视图的样式，如果是 month 默认是一个月

day

week

month



练习 2 - 日历视图
添加一个日历视图对象只允许用户蓝色的信息

7.3 查询视图

搜索视图可以绑定到指定的窗口事件中使用 search_view_id 进行关联，这样同一个对象可以有不同的搜索方法

定义搜索视图时你可以使用 group, separator, label, search, filter, newline 属性

过滤器可以定义成按钮，添加一个上下文属性字段

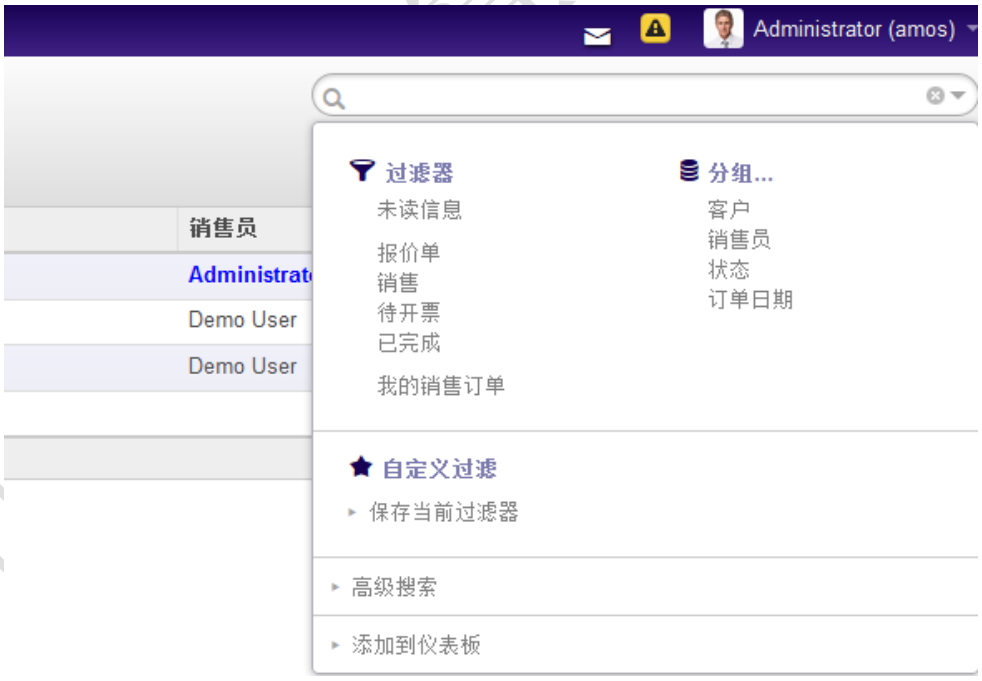
```
<search string="Ideas">
        <filter name="my_ideas" domain="[('inventor_id','=',uid)]" string="My
    Ideas" icon="terp-partner"/>
```

```
        <field name="name"/>
        <field name="description"/>
        <field name="inventor_id"/>
        <field name="country_id" widget="selection"/>
    </search>

    <!-- filter: 最近三个月 -->
      <filter icon="terp-personal"
            name="last_three_month"
            string="Last 3 Months"
            domain="[('date','&lt;=',time.strftime('%%d/%%m/%%Y')),
            ('date','&gt;=',((context_today() -relativedelta(months=3)).strftime('%%d/%%m/%%Y')))
           ]"/>
```

解释:

time.strftime('%%d/%%m/%%Y') 返回的是当前日期; <= 是小于号, 在 XML 中只能这样表示. 这一行表示: "在今天之前", context_today() 是 Odoo 中另外一种返回当前日期的方式, 它减去 relativedelta(months=3) 就是三个月前. 这三行表示: "大于三个月前", 其中 >= 是 XML 中的大于号, 合起来表示"三个月前到今天", 即"过去三个月"



打开上图的搜索，你可以使用高级查询，上下文的值是一个 Python 字典

- default_foo：查询表单字段.
- search_default_foo 增加一个条件再进行搜索

练习 3 - 查询视图

添加搜索视图包含：

1）要搜索的字段//会议名称//的基础上他们的头衔

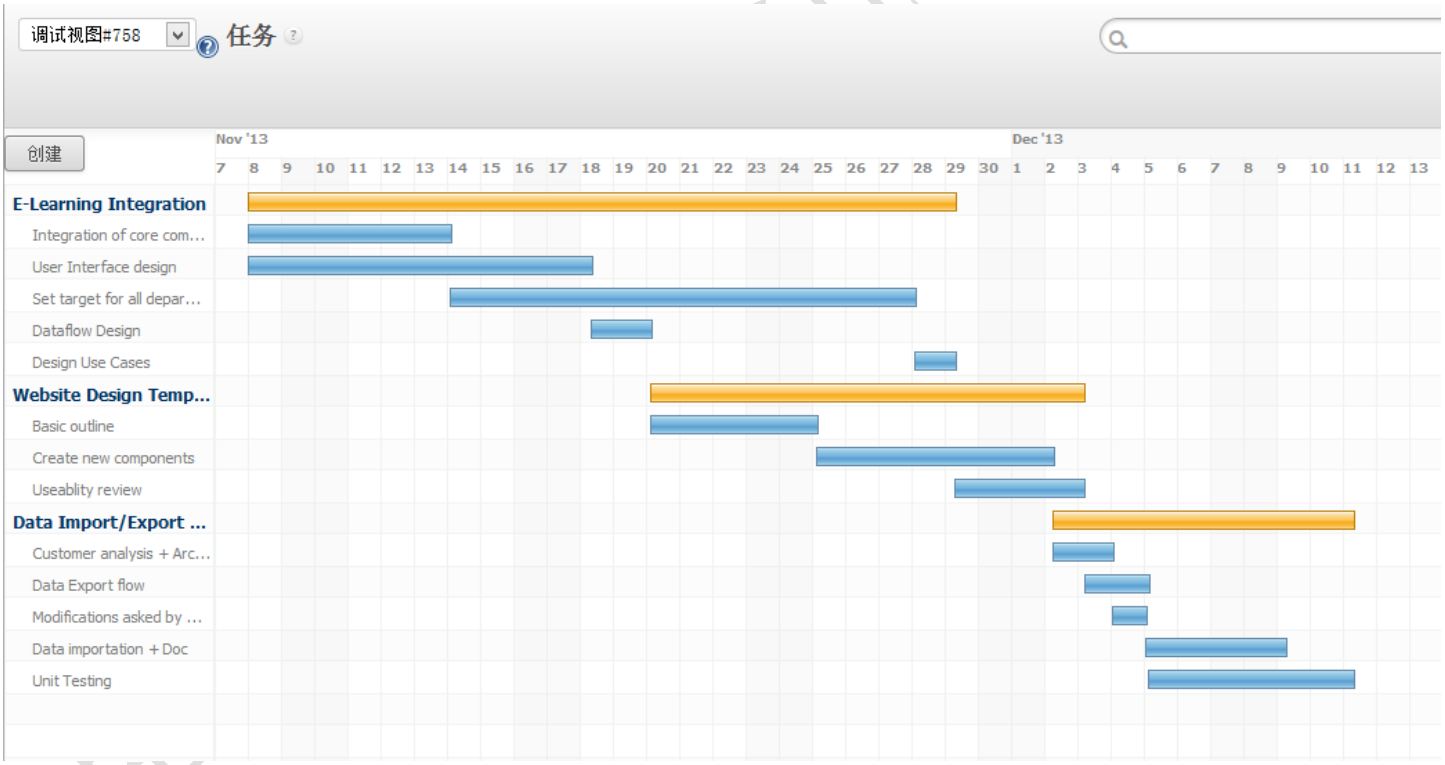
2）一个按钮来过滤当前用户是负责的会议

7.4 甘特图

条形图，通常用于显示项目进度

属性	与日期视图一样
字段属性	定义甘特图水平方向的字段，根据日期，等级不同向下进行排列

```
<gantt string="Ideas" date_start="invent_date" color="inventor_id">
    <level object="idea.idea" link="id" domain="[]">
        <field name="inventor_id"/>
    </level>
</gantt>
```



练习 4 - 甘特图

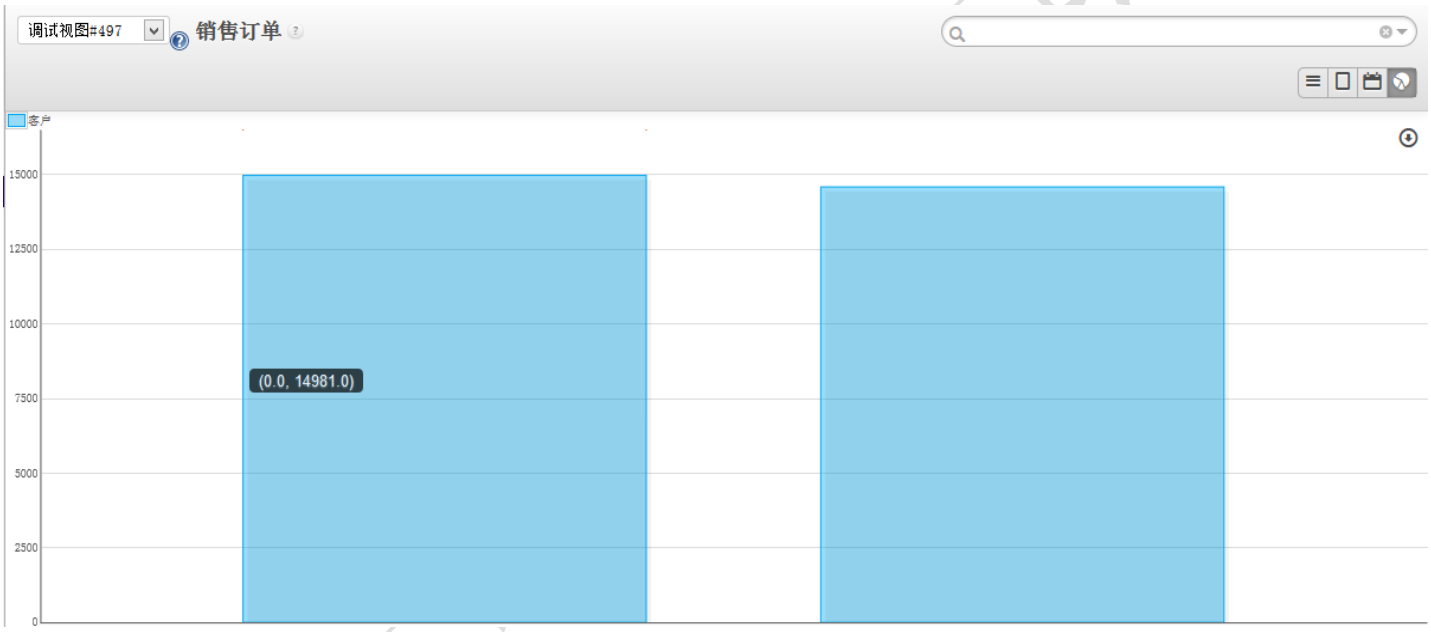
添加甘特图，使用户能够查看的会议安排。会议应安负责人进行分组

7.5 图表

视图，用于显示统计的图表（<graph>父元素）。

属性	type: bar, pie (default)　类型：条形图，饼图（默认）
	orientation: horizontal, vertical　方向：水平，垂直
字段参数	图型有三个轴 X ， Y， Z
	X ， Y 是必须，Z 是可以不写
	分组时有下面几种合计方法(+,*,**,min,max)

```
<graph string="Total idea score by Inventor" type="bar">
    <field name="inventor_id" />
    <field name="score" operator="+"/>
</graph>
```



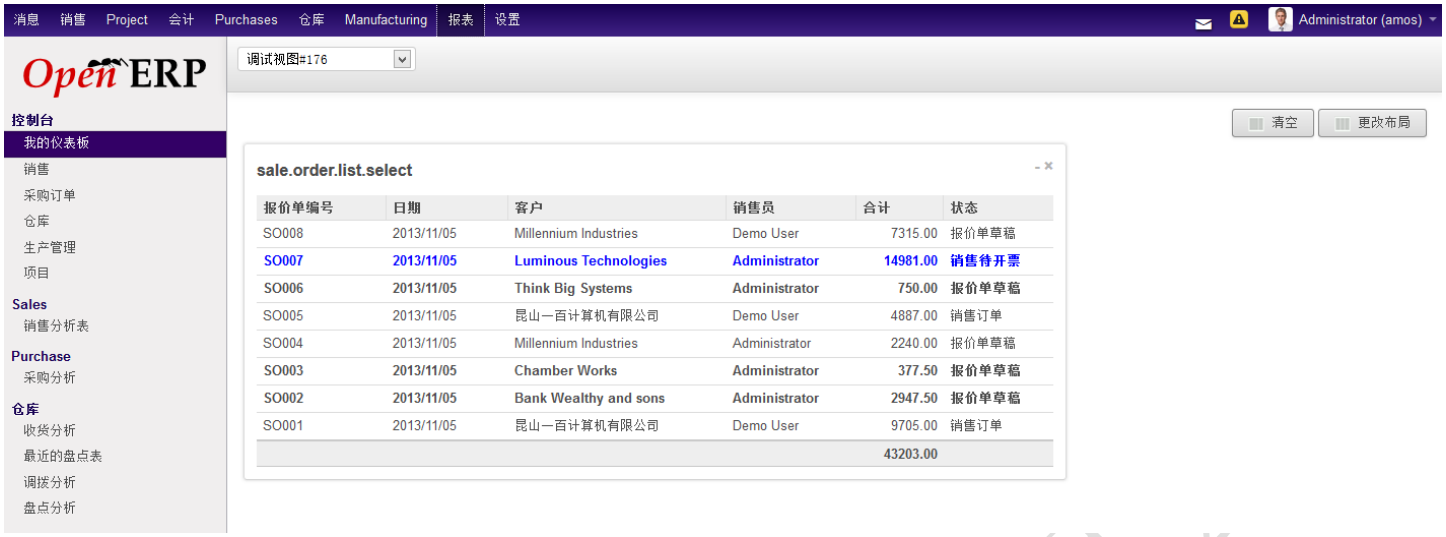
练习 5 - 图表视图

添加一个图形视图显示，每场会议有多少人参加

7.6 仪表盘

练习 6 - 定义仪表盘

通过会议模块创建的日历视图、列表视图、统计图（可切换到一个表单视图）。并给这个视图创建一个菜单



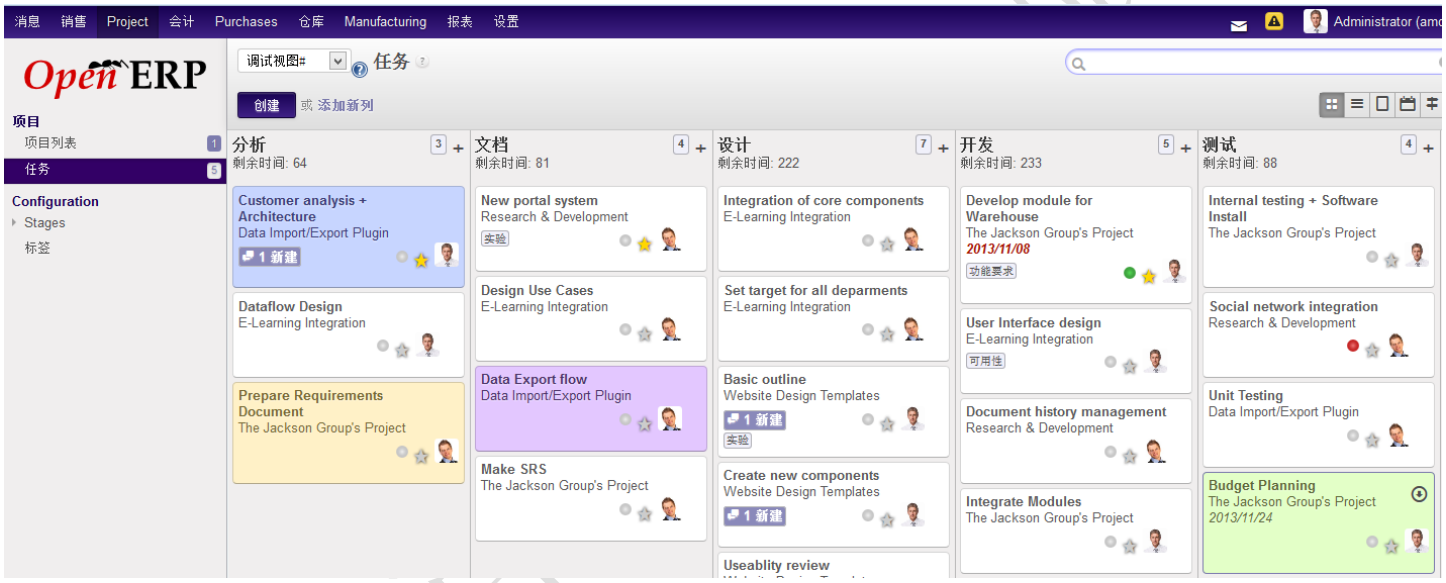
7.7 看板

每一个组都代表一条记录，可以查看线索，支持 Html 编码

```
<record model="ir.ui.view" id="view_openacad_session_kanban">
    <field name="name">openacad.session.kanban</field>
    <field name="model">openacademy.session</field>
    <field name="type">kanban</field>
    <field name="arch" type="xml">
      <kanban default_group_by="course_id">
<field name="color"/>
        <templates>
          <t t-name="kanban-box">
            <div t-attf-class="oe_kanban_color_#{kanban_getcolor(record.color.raw_value)} oe_kanban
              <div class="oe_dropdown_kanban">

                <!-- dropdown menu -->
                <div class="oe_dropdown_toggle">
                  <span class="oe_e">í</span>
                  <ul class="oe_dropdown_menu">
                    <li><a type="delete">Delete</a></li>
                    <li><ul class="oe_kanban_colorpicker" data-field="color"/></li>
                  </ul>
                </div>
                <div class="oe_clear"></div>
```

```
            </div>
            <divt-attf-class="oe_kanban_content">
                <!-- title -->
                Session name : <field name="name"/> <br/> Start date : <field
            name="start_date"/><br/> duration : <field name="duration"/>
            </div>
        </t>
    </templates>
</kanban>
</field>
</record>
```



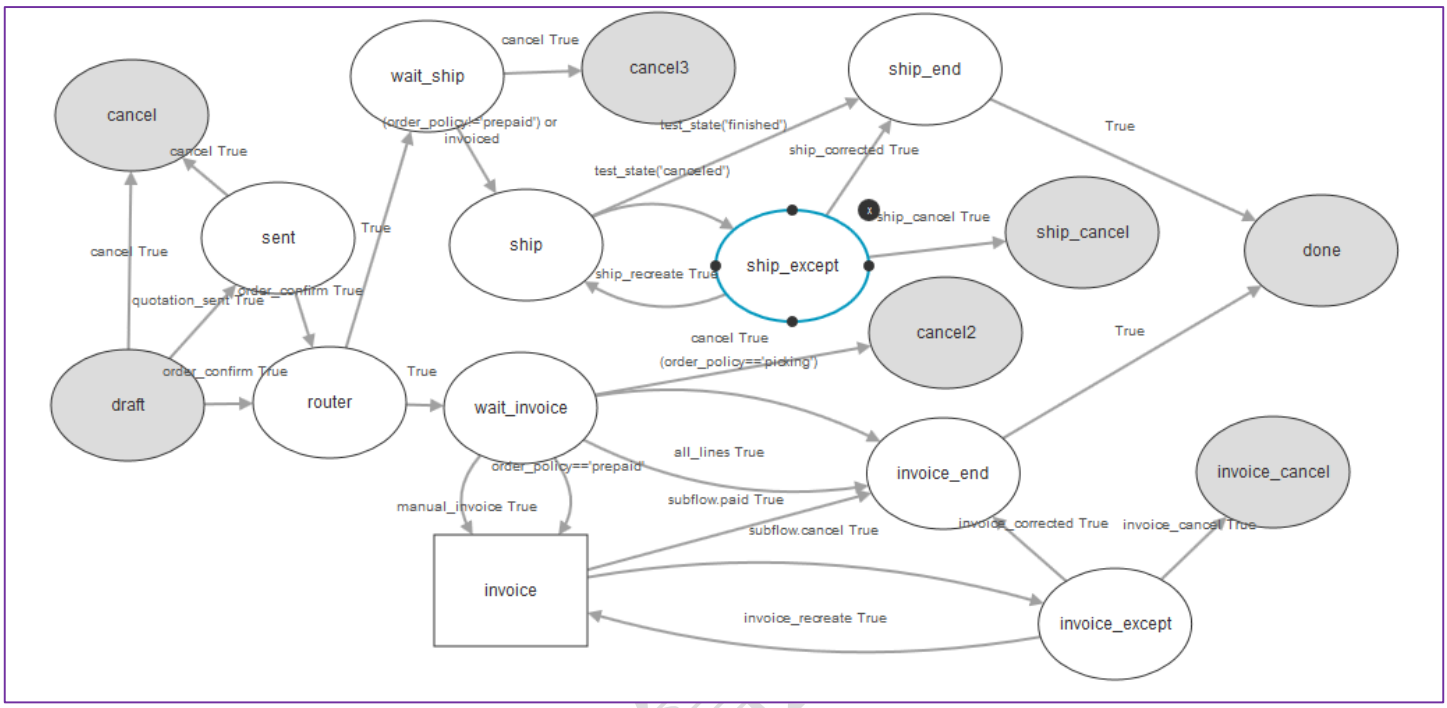
练习 7 - 看板视图

添加看板显示会议,使用用列会议进行分组

第8章 工作流

工作流是描述他们的动态模型运动规律。随着时间的推移工作流还可用于追踪过程。

工作流和工作流实例：工作流定义了对某一类型的对象，如采购订单（PO）的处理流程。



例如，PO 单的一般处理流程也许是：

1）新建 PO，State = draft；

2）审批 PO，审批的同时，

a)系统自动产生仓库收货单；

b)系统自动产生凭据（Invoice），供财务确认付款；

c)系统自动产生 PDF 的采购订单，并自动 EMail 给该 PO 单对应的供应商。

但对于特定的某个 PO 对象，需要一个工作流实例，以记录本 PO 对象处在流程的哪个阶段，如 PO1 尚在 draft 状态，PO2 已经审批通过。

PO 单的审批，以及对应的 a)、b)、c)的动作，都可以在 OE 的工作流中定义解决，而不需要全编码在 PO 对象上。即工作流实现了流程处理相关的代码和被处理对象的代码相分离，降低了不同处理代码的耦合性，增加了系统功能的柔软性。

8.1 工作流定义：

```
<?xml version="1.0"?>
  <terp><data>
    <record model="workflow" id=workflow_id>
    <field name="name">workflow.name</field>
    <field name="osv">resource.model</field>
```

```
    <field name="on_create">True | False</field>
  </record>
</data></terp>
```

model：固定取值"workflow"

id：任意值，唯一标识本工作流

name: 工作流的名称，任意定义

osv： 本工作流关联的对象类型，是 Odoo 模块中定义的某对象名，如采购单对象（purchase.order）。是本工作流处理的数据对象。

on_create：每当系统新产生一个 osv 中定义的对象的实例时候，是否对应的产生一个和该对象实例关联的工作流实例。默认是 True.

8.2 活动（Activity）定义

```
<record model="workflow.activity" id="activity_id">
  <field name="wkf_id" ref="workflow_id"/>
  <field name="name">activity.name</field>
  <field name="kind">dummy | function | subflow | stopall</field>
  <field name="subflow_id">subflow_id</field>
  <field name="action">(...)</field>
  <field name="action_id">(...)</field>
  <field name="split_mode">XOR | OR | AND</field>
  <field name="join_mode">XOR | AND</field>
  <field name="signal_send">(...)</field>
  <field name="flow_start">True | False</field>
  <field name="flow_stop">True | False</field>
</record>
```

model：固定取值 workflow.activity

wkf_id：本 Activity 所属的工作流 id

name: 本 Activity 名称，任意值

kind：本 Activity 类型，有 Dummy, Function, Subflow, Stop All 四种。kind 说明，如果流程到达本节点，系统应执行的动作类别。

Dummy 表示不执行任何动作，即 action 中定义的代码不会被执行。

Function 表示执行 action 中定义的 python 代码，且，执行 action_id 中定义的 server action。常见情况是，action 中定义一个 write 方法，修改流程关联的对象的状态。对于 Function 类型的节点，action 中定义的代码或者返回 False，或者返回一个客户端动作 id（A client action should be returned）。

Subflow 类型表示触发“subflow_id”中指定的工作流。仔细的读者或许要问，工作流的执行总是和某个被处理的对象关联，是的，如果定义了 action，subflow 关联的对象 id 由 action 中定义的代码返回。如果没有定义 action，系统默认 subflow 关联的对象和本节点所属的工作流处理的对象 id 一致。stopall 类型表示，流程到此节点则结束，但结束前，系统仍会执行 action 中的代码。

signal_send：执行完本节点的动作（action 及 action_id 定义的动作）后，应向别的工作流发往的 signal，格式是：subflow.signal。subflow_id 和 signal_send 必须配合使用，subflow_id 表示，触发子工作流 subflow_id，在该子工作流中，通常必须定义 signal_send，signal_send 定义父流程中的某个 signal，表示，子流程处理结束后触发父流程中的信号 subflow.signal。注意，用于父子流程通信的工作流 signal 必须是形如 subflow.* 。例如，在 HR 模块的 workflow "wkf_expenses"中，需要开发票时候，它触发流程 account 模块中的工作流 “account.wkf”（ <field name="subflow_id" ref="account.wkf"/> ）。account.wkf 处理完成后，发出信号 subflow.paid 通知 wkf_expenses 流程（ <field name="signal_send">subflow.paid</field> ）。wkf_expenses 中定义了信号 subflow.paid（ <field name="signal">subflow.paid</field> ）。

split_mode：有三个选项，XOR，OR，AND，默认是 XOR。XOR 表示，由本节点始发的出迁移中，沿着第一个满足迁移条件的迁移跳转。OR 表示由本节点始发的出迁移中，只要满足迁移条件即沿该迁移跳转。AND 表示由本节点始发的出迁移中，只有所有迁移皆满足迁移条件才跳转，而且是同时沿所有迁移跳转。XOR 只有一个跳转，OR 有零或多个跳转，AND 有零或全部跳转。

join_mode：有两个选项，XOR，AND，默认是 XOR。XOR 表示，以本节点为终点的入迁移中，只要有一个跳至本节点，即执行本节点的 action。AND 表示，以本节点为终点的入迁移中，只有所有迁移都已经跳至本节点，才执行本节点的 action。

flow_start：表示流程的开始节点。

flow_stop：表示流程的结束节点。

8.3 迁移（Transition）的定义

迁移的完整 XML 定义格式如下。

```
<record model="workflow.transition" id="transition_id">
  <field name="act_from" ref="activity_id_1"/>
  <field name="act_to" ref="activity_id_2"/>
  <field name="signal">(...)</field>
  <field name="condition">(...)</field>
  <field name="trigger_model">(...)</field>
  <field name="trigger_expr_id">(...)</field>
</record>
```

act_from：本迁移的起始节点，引用之前定义的 Activity。

act_to：本迁移的结束节点，引用之前定义的 Activity。

signal：触发本迁移的信号，表示，如果系统收到 signal 定义的信号，则触发本迁移。触发信号有三种方式，1）最常见的是用户点击视图中的“name = 本处定义的 signal”的 button，此时相当于向系统发送迁移信号量。系统会根据视图中的对象 id，找到对象关联的 workflow，再找到与 button name 相同的 signal，触发之。2）调用 workflow_service 的方法：trg_validate(self, uid, res_type, res_id, signal, cr)，此方法表示，触发对象类型 res_type 关联的 workflow 的 signal 信号，工作流实例关联的对象实例是 res_id。3）子流程的 signal_send 发出的信号，此种情况前文已说过。

condition：迁移的条件，是一段 Python 代码，通常是一个函数调用。当系统收到 signal 中定义的信号时候，检查此处的条件，条件为真则实际触发迁移。

trigger_model 和 trigger_expr_id：此二字段表示启动一个新工作流实例。trigger_model 定义对象类型，trigger_expr_id 定义一段 Python 代码，返回 trigger_model 类型的对象 id。此二字段表示，如果 act_from 中的 action 执行完毕，且 condition 条件 OK，则系统中插入一个 trigger_model 类型，trigger_expr_id 返回的对象 id 关联的工作流实例。然后，可以调用 workflow_service 的方法 trg_trigger(self, uid, res_type, res_id, cr)实际执行该工作流。实际使用例子请参考 Sale 模块的工作流定义 wkf_sale：

```
<field name="trigger_model">procurement.order</field>
<field name="trigger_expr_id">procurement_lines_get()</field>
```

« 最后编辑时间: 六月 27, 2011, 03:31:46 下午 作者 NewZN »

迁移（Transition）的定义漏了权限组 group_id，修正如下。表示只有该权限组可以触发本迁移。

迁移的完整 XML 定义格式如下。

```
<record model="workflow.transition" id="transition_id">
  <field name="act_from" ref="activity_id_1"/>
  <field name="act_to" ref="activity_id_2"/>
  <field name="group_id" ref="groupid"/>
  <field name="signal">(...)</field>
  <field name="condition">(...)</field>
  <field name="trigger_model">(...)</field>
  <field name="trigger_expr_id">(...)</field>
</record>
```

split_mode 与 join_mode 举例呢

例如 split_mode 的三个选项，XOR，OR，AND。以办公审批流程中的“会签”为例，XOR 表示，所有审批人中，只要一个人审批了，就表示审批通过。AND 表示，只有所有人都审批了，才表示审批通过。

Trigger model 和 Trigger expr_id 的解释不够清楚，更清楚的解释参看链接：

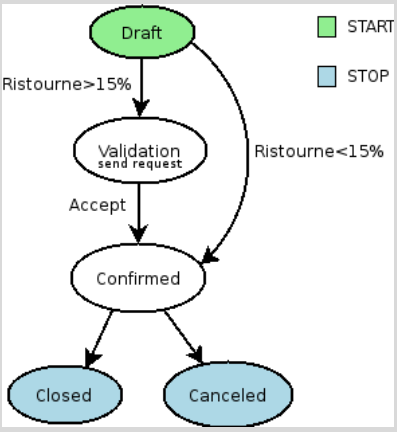
http://www.Odoo.com/forum/post82154.html#p82154

练习 1 - 静态工作流程

添加单据的状态类型，草稿，等待处理，完成:,使用按钮来完成。

```
'state': fields.selection([('draft', u'草稿'),
                           ('running', u'等待处理'),
                           ('done', u'完成'), ],
                          u'状态', ),
```

- 草稿→等待处理
- 等待处理→草稿
- 等待处理→完成
- 完成→草稿



我们来看一个 Odoo 的例子下面是一个销售订单生成发票的工作流。

Odoo 的工作流是完全可以定制的，工作流主要是用来管理业务对象的生命周期，通点节的条件的转换，也可以使用图型化界面来设计，也可以使用 XML 来声明各个节点之间的关联，有开始，节点到节点，最后到完成通常叫工作流．

练习 2 - 动态工作流编辑器

使用在线图型工作流编辑器来设计上面静态工作流

练习 3 - 自动转换

草稿转为等待处理时要判断参加会议的人数要大于坐席人数一半

练习 4 - XML 工作流程

安装模块 base_module_record。用它来导出到刚才在图型界面上设计的工作流，您可以放置在你的模块相关模块中

练习 5 - 服务器操作

使用服务动作来自动触发事件，同样保存时如果参加会议的人数大于坐席的一半就自动确认到等待处理状态

第9章 安全

必须配置的访问控制机制，实现了一致的安全政策，权限管理有四个层次

- **菜单级别：** 即，不属于指定菜单所包含组的用户看不到该菜单。不安全，只是隐藏菜单，若用户知道菜单 ID，仍然可以通过指定 URL 访问
- **对象级别：** 即，对某个对象是否有‘创建，读取，修改，删除“的权限。系统中的对象可以简单理解为表对象，比如“客户”，“产品”，“销售订单”等都是对象
- **记录级别：** 即，对对象表中的数据的访问权限。比如同样访问“客户”对象，业务员只能对自己创建的客户有访问的权限，而经理可以访问其所辖的业务员的所有“客户”对象，这里的访问也可以进一步明细到“创建，读取，修改，删除”的权限
- **字段级别：** 即，一个对象或表上的某些字段的访问权限。比如产品的成本字段只有经理有读权限，比如订单上的单价字段只有经理才有修改的权限等。

在实际的培训和实施过程中，我们发现客户往往会提出很多记录级别的访问规则要求。最经典的就是：“我希望销售员只能看到他自己创建的客户，而其经理则可以看到所有业务员创建的客户信息”，本文就是以这个需求为蓝本来介绍如何使用“记录规则”来定制记录级别的访问控制。

Odoo 权限的核心是权限组（res_groups）。对每个权限组，可以设置权限组的 Menus，Access Right，Record Rule。

Menus 表示，该权限组可以访问哪些菜单。如果指定某权限组可以访问某父菜单，那么，系统会根据该权限组可访问的对象（Access Right 中定义）自动计算，哪些子菜单可以显示。计算规则是，如果没有为该子菜单指定任何权限组，且该权限组对该子菜单关联的对象有至少读的权限，那么，系统会自动显示该菜单。如果不希望系统自动显示某子菜单，只要把该子菜单加入系统自带的“Useability / No One”权限组，该菜单就不会被显示了。“Useability / No One”通常用来隐藏某些菜单，通常不会指定任何用户属于“Useability / No One”权限组。

Access Right 表示，该权限组可以访问哪些对象，以及拥有读、写、删、建中的哪个权限。如下图中最后一行，表示，Employee 权限组对 Partner（res_partner）对象只有读权限。

Record Rule 表示，该权限组可以访问对象中的哪些记录，以及拥有读、写、删、建中的哪个权限。Access Right 指定的权限，是对该对象的数据表里的所有记录拥有该权限。Record Rule 指定，只对该对象的数据表里的某些记录（通过定义过滤条件 Domain 指定）拥有某些（读、写、删、建）权限。

如上图表示，Employee 权限组，对申购单（Purchase Requisition） 对象，对本部门的，且处于草稿状态的申购单，拥有创建、删除、更新的权限。对于非本部门的或者不是草稿状态的申购单，由于不符合 Domain 条件，更新或删除时候，系统都会报错。

['&',('department', '=', user.context_department_id.id),('state', '=', 'pr_draft')]

这个 Domain 条件表示，申购单的部门等于当前用户的部门，且申购单的状态是草稿（pr_draft）。系统的实际实现是，在数据库访问的 Update， Delete 等语句中，强行加上本处定义的 Domain 条件（因此在系统内部，此处的 Domain 条件叫“Domain_Force”，哈哈）。

字段权限，还可以指定，某字段只能供某权限组访问。Access Right 和 Record Rule 表示，权限组可以访问哪些对象，以及对象里的哪些记录。而字段权限指定，权限组能访问记录里的哪个字段。如下例表示，只有 base.group_admin 权限组才可以读、写 name 字段。

‘name’ : fields.char(‘Name’ , size=128, required=True, select=True, write=['base.group_admin'], read=['base.group_admin']),

又如下例在视图上指定，只有 group_product_variant 权限组才能看到产品的 variants（规格）字段。

工作流权限，在工作流的迁移（Transition）的定义中，可以指定哪个权限组可以触发本迁移，定义语法是： 。

灵活组合上述权限设置，可以满足非常复杂的权限要求，如工作流的审批权限，菜单的访问权限，记录的访问权限，字段的访问权限，等等。

9.1 基于组的访问控制机制

创建一个权限组对象是“res.groups”，即没有给你菜单有些对象你还是可以间接访问，所以实际对象级权限是 (read, write, create, unlink)，定义这些访问对象通常是通过 CSV 文件导入的，另外也可以使用字段组的属性视图来限制特定字段的访问权限。

9.1.1 groups_id 的使用

- 某些行对于某些权限组的人是可写入，但是其他权限组的人是只读。

例如我新建一个视图 view1 继承产品视图 view 修改成本价为只读，view2 修改成本价可写，groups_id 是采购经理，那么没有采购经理权限的人打开视图看到的是成本价只读（view+view1），采购经理看到的是可写的（view+view1+view2）。

- 具有某些权限组的人看到的视图更丰富。（例如：Odoo 里面销售订单行 editable 的设置就是通过 groups_id 来实现）

```
<!--sale/sale_view.xml-->
        <record id="view_order_form_editable_list" model="ir.ui.view">
            <field name="name">sale.order.form.editable.list</field>
            <field name="model">sale.order</field>
            <field name="inherit_id" ref="sale.view_order_form"/>
            <field name="groups_id" eval="[(6, 0, [ref('product.group_uos'), ref('product.group_stock_packaging'),
ref('sale.group_mrp_properties')])]"/>
            <field name="arch" type="xml">
                <xpath expr="//field[@name='order_line']/tree" position="attributes">
                    <attribute name="editable"/>
                </xpath>
            </field>
        </record>
```

这里就是上面说的 2 里面的实现方式，这里为视图 view_order_form_editable_list 定义了 groups_id，我们来一起分析下。

eval:把 eval 的值通过作为 python 运算后返回到该属性，这里就是 eval 后的值会返回给 groups_id。

ref:Odoo 视图的方法。根据 module_name.xml-id 返回数据库 id。

[(6, 0, [xx,yy])]: 看这里 http://cn.Odoo.cn/Odoo_import_image_by_xmlrpc/

9.2 访问权限

权限对象模型是 ir.model.access.csv”访问权限关联到一个模型，每一个模型都有，增，删，改，查，通常建立权限都是使用 CSV 来导入

id,name,model_id/id,group_id/id,perm_read,perm_write,perm_create,perm_unlink access_idea_idea,idea.idea,model_idea_idea,base.group_user,1,1,1,0

access_idea_vote,idea.vote,model_idea_vote,base.group_user,1,1,1,0

访问权限属性
id ：权限 ID 不可重
name ：描述
model_id/id：对象
group_id/id：组的名称
perm_read：只读
perm_write：修改
perm_create：创建
perm_unlink:删除

练习 1 - 通过 Odoo 的接口访问控制
创建一个用户叫 amos 再创建一个用户组 把 Amos 加入到当前组中

练习 2 – 通过 XML 添加访问控制模块
定义 XML 来创建两个组，一个用户组，一个经理组

练习 3 – 通地 CVS 导入访问控制权限
使用 CSV 添加，读。写，创建，删除对象到用户组全部，一个经理组全部

9.3 记录规则

A 记录规则的访问权限限制给定的模型记录的一个子集。规则是一个记录模型“ir.rule”，并关联到一个模型，一组数（many2many 场），权限的限制适用，和域。的域指定记录的访问权限是有限的。

下面是一个例子的规则，以防止删除的线索，是不是“取消”状态。请注意，必须遵循的价值领域的“群体”的约定相同的方法“写”的 ORM。

```
<record id="delete_cancelled_only" model="ir.rule">
    <field name="name">Only cancelled leads may be deleted</field>
    <field name="model_id" ref="crm.model_crm_lead"/>
    <field name="groups" eval="[(4, ref('base.group_sale_manager'))]"/>
    <field name="perm_read" eval="0"/>
    <field name="perm_write" eval="0"/>
    <field name="perm_create" eval="0"/>
    <field name="perm_unlink" eval="1" />
    <field name="domain_force">[('state','=','cancel')]</field>
</record>
```

练习 4 - 记录规则

添加一条规则到“OpenAcademy / Manager”，不许写入和删除，如果会议没有负责人，该组的所有用户必须都可以修改它

第10章　向导

10.1 向导对象（osv.TransientModel)

向导是一个动态与用户进行互动对话的模型，向导使用“osv.TransientModel”而不是其它 “osv.Model”.osv.TransientModel”扩展出“osv.Model” 重用现有方法，具有特殊功能

1. 向导的数据不是固定的，系统会定期清除数据。
2. 向导是没有任何权限限制，用户可以随意创建。

向导的字段定义与常规则字段定义一样，但是常规模块不能指定 many2one 到向导。

练习 1 - 定义向导类
创建一个向导模式（继承 osv.TransientModel）使用 many2one 关系与当前 Session 对象和 one2many 的参会对象关联

备注:key2 几个固定参数
向导：client_action_multi
报表：client_print_multi
栏相关链接：client_action_relate
双击树视图：tree_but_open
tree_but_action：已弃用同 tree_but_open

10.2 向导实例

向导是通过”ir.actions.act_window”来触发，你可以修改 target 属性弹出多种样式。

还有一种使用方法启用向导同样也是使用”ir.actions.act_window”但是多了一个 src_model 用来指定上下文，这里使用了系统固定的 key2 来显示菜单在那里出现

```
<act_window
    id="session_create_attendee_wizard"
    name="Add Attendees"
    src_model="context_model_name"
    res_model="wizard_model_name" view_mode="form"
    target="new"
    key2="client_action_multi"/>
```

向导属性 act_window

id 不可以重名
name 菜单名称
src_model 上级对象
res_model 向导对象
view_mode 视图类型
target 打开方式
key2 显示位置见详细固定参数说明
例：
<button string="Name" name="%(session_create_attendee_wizard)d" states="running" type="action" colspan="1"/>

Target 属性
current 当前窗口
new 新建窗口
inline 行内编辑
inlineview 内嵌视图

练习 2 - 创建一个向导菜单
使用向导创建一个菜单并执行一个弹出事件

10.3 向导视图

常规向导用法有两个确认事件，选择取消后窗口自动消失

```
<button string="Do it!" type="object" name="some_action"/>
<button string="Cancel" special="cancel"/>
```

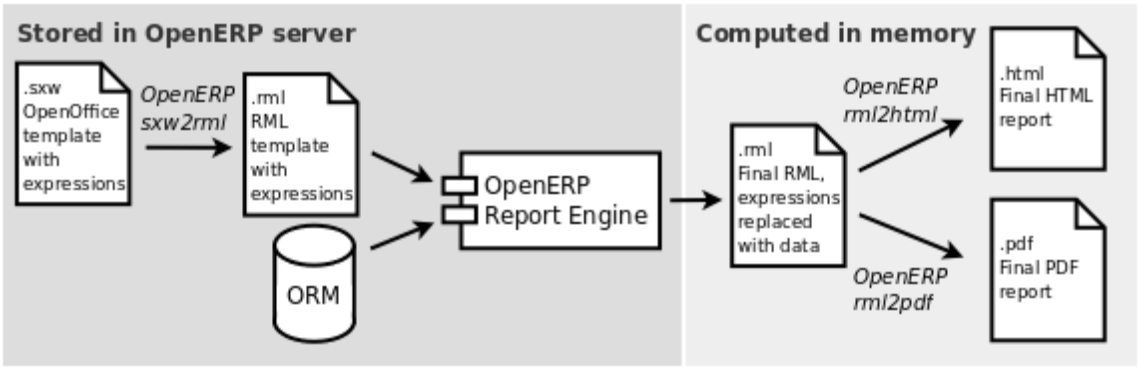
练习 3 - 自定义表单视图
定制表单视图以显示所有的类的字段

练习 4 - 创建方法
创建方法 action_add_attendee 类 CreateAttendeeWizard 中，实现它，并添加一个按钮，表单视图中调用它。同时添加一个“取消”按钮，关闭向导窗口

练习 5 - 绑定上下文

绑定上下文信息模型向导。

提示：使用“上下文”来定义当前会话的 session_id 默认值



练习 6 - 向导添加多个记录

请向导可以添加一次参加几次会议

第11章 多语言

每个模块都可以提供自己的翻译文件，存放在 i18n 的目录，文件名为 LANG.po 其中 LANG 是语言，语言环境的代码或语言和国家组合时，它们之间的区别（如 pt.po 或 pt_BR.po ）。会被自动加载 Odoo 的所有启用的语言翻译。开发者总是使用英语创建一个模块时，再使用 Odoo 的 gettext 翻译模块 POT 导出翻译模块（设置>翻译>导出一个翻译文件没有指定的语言），创建的模块 POT 文件，然后得出的翻译 PO 文件。许多 IDE 插件支持编辑 PO/ POT 文件和合并。

提示:

Odoo 的使用 GNU gettext 的格式（可移植对象），可以多人在线协同翻译。

```
|- idea/                #模块
|- i18n/                # 翻译文件
    | - idea.pot        # 翻译模板(Odoo)
    | - fr.po           # 法语翻译
    | - pt_BR.po        # 巴西葡萄牙语翻译
    | (...)
```

提示:

默认情况下 Odoo 的翻译仅抽取标签内的 XML 文件或内部字段定义在 Python 代码,但任何 Python 字符串可以被翻译这种方式通过工具翻译。_方法(如。_('标签'))

11.1 各个标签翻译方法

11.1.1 单据状态翻译

```
#. module: amos_template
#: field:amos.template,state:0
msgid "Status"
msgstr "状态"
```

11.1.2 界面按钮翻译

```
#. module: amos_template
#: view:amos.template:0
msgid "Review"
msgstr "提交上级"
```

11.1.3 数据库名称翻译

```
#. module: amos_template
#: model:ir.model,name:amos_template.model_amos_template
msgid "Template"
```

msgstr "表名翻译"

11.1.4 字段翻译

#. module: amos_template
#: field:amos.template,partner_id:0
msgid "Customer"
msgstr "客户"

11.1.5 字段帮助翻译

#. module: amos_template
#: help:amos.template,name:0
msgid "Document Number."
msgstr "一般为自动生成的编号"

11.1.6 菜单翻译

#. module: amos_template
#: model:ir.ui.menu,name:amos_template.menu_amos_template
msgid "Template"
msgstr "模板"

11.1.7 事件窗体翻译

#. module: amos_template
#: model:ir.actions.act_window,name:amos_template.action_amos_template
msgid "Template Demo"
msgstr "窗体界面"

11.1.8 代码片段翻译

#. module: amos_template
#: code:addons/amos_template/amos_template.py:125
#, python-format
msgid "You can only delete draft!"
msgstr "你只能删除草稿单!"

11.1.9 数据库字段约束

#. module: amos_template
#: sql_constraint: amos.template:0
msgid "Order Reference must be unique per Company!"
msgstr "订单号必须在一个公司范围内唯一"

11.1.10 窗体 XML 帮助翻译

#. module: amos_template
#: model:ir.actions.act_window,help:amos_template.action_amos_template
msgid ""
"A brief overview of the features of this module."
msgstr ""
"简单的概述这个模块实现的功能"

11.1.11 搜索翻译

```
#. module: amos_template
#: view:amos.template:0
msgid "Unread Messages"
msgstr "未读消息"
```

11.1.12 搜索帮助翻译

```
#. module: amos_template
#: view: amos.template:0
msgid "Order that haven't yet been confirmed"
msgstr "订单尚未被确认"
```

11.1.13 弹出窗体提醒翻译

```
#. module: amos_template
#: view:amos.template.warning:0
msgid "This guy is lazy nothing left."
msgstr "这家伙很懒什么也没有留下."
```

11.1.14 界面视图翻译

```
#. module: amos_template
#: view:amos.template:0
msgid "Verify"
msgstr "审核"
```

11.1.15 权限分类组翻译

```
#. module: amos_template
#: model:ir.module.category,name:amos_template.base_module_amos_template
msgid "Template"
msgstr "模板"
```

11.1.16 权限组翻译

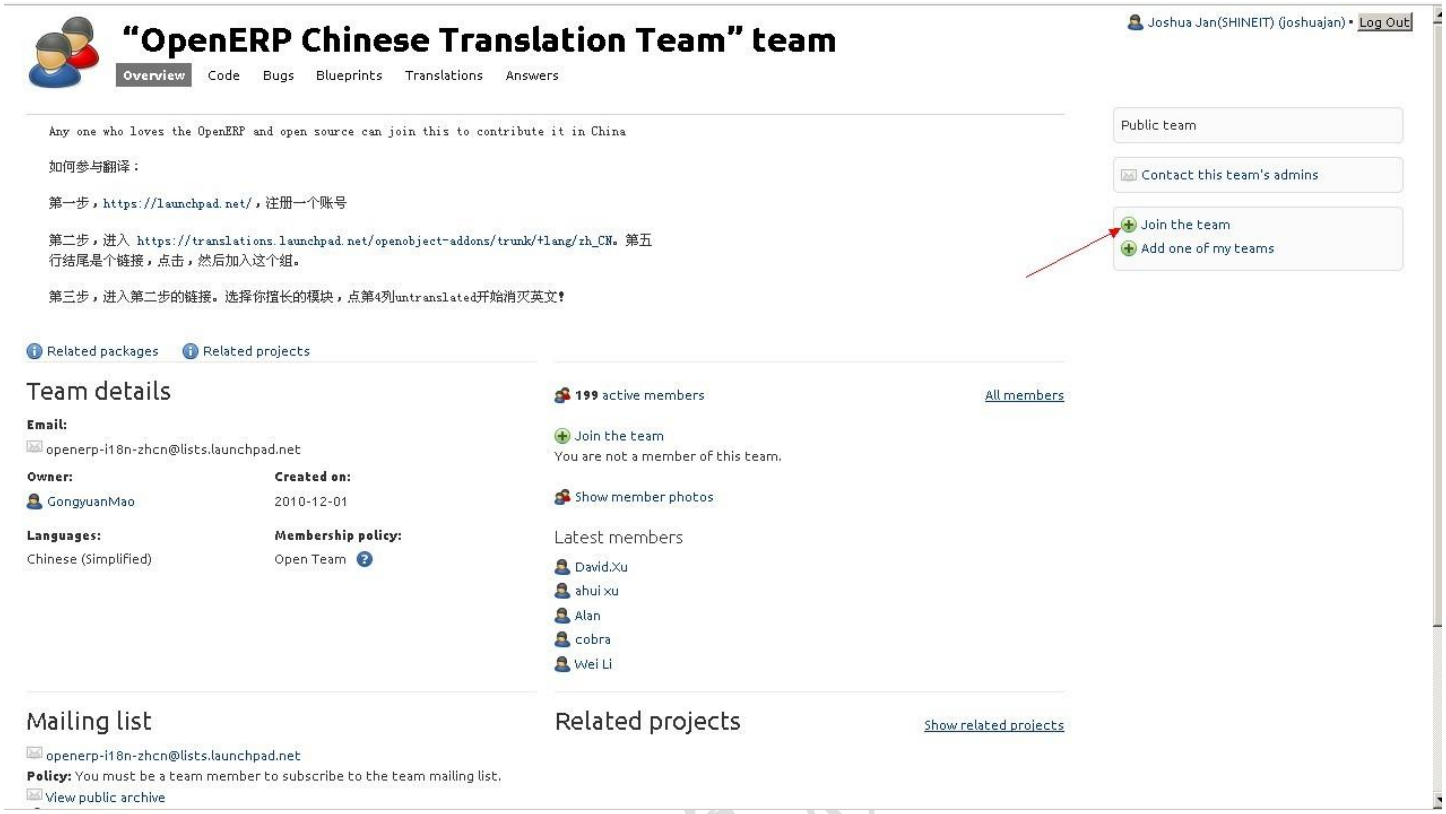
```
#. module: amos_template
#: model:res.groups,name:amos_template.group_amos_template_user
msgid "Template User"
msgstr "模板用户组"
```

11.1.17 插入带的翻译数据

```
#. module: amos_template
#: model:amos.template.category,name:amos_template.amos_template_category_2
msgid "Internal"
msgstr "内部"
```

练习 1 - 翻译模块
选择安装 Odoo 的第二语言。翻译你的模块提供的 Odoo 使用的设施

11.2 加入 Odoo 的中文翻译



1. 你只需要注册一个 https://login.launchpad.net/+login 的账号
2. 加入翻译组
3. .翻译！
   addons: https://translations.launchpad.net/openobject-addons/7.0/+lang/zh_CN
   server: https://translations.launchpad.net/openobject-server/7.0/+pots/base/zh_CN/+translate
   web: https://translations.launchpad.net/Odoo-web/7.0/+lang/zh_CN
   文档: https://translations.launchpad.net/openobject-doc/7.0/+lang/zh_CN

第12章 报表

12.1 打印报表

Odoo 报表有多种方式实现:

- **自定义报表**: 这些报告可以直接通过客户端界面创建，无需编程。这些报告表示业务对象（ir.report.custom）

高品质报告，使用 OpenReport 方法 无需编程，但你必须写 2 个小型的 XML 文件:

1. 报表模板
2. XSL：RML 样式表

- **WebKit 引擎**: 基于 Mako HTML 模板(http://www.makotemplates.org) 生 成 PDF(wkthtmltopdf). 你可以从(http://www.camptocamp.com)这里下载。
- 硬编码的报告
- **OpenOffice**

Odoo 的报告引擎有几个，从不同的来源，以及多格式生成报告。

12.2 自定义报表

12.3 WebKit 报表

安装 webkit 使用模板来设计报表，你需要定义一个报表 XML 如下：

```
<report
   string="WebKit invoice"
   id="report_webkit_html" model="account.invoice"
   name="webkitaccount.invoice"
   file="report_webkit_sample/report/report_webkit_html.mako"
   webkit_header="report_webkit.ir_header_webkit_basesample0"
   report_type="webkit"
   auto="False"/>
```

这个例子是来自模块报表 webkit 实例。打开 report_webkit_html.mako ，可以 html 语法来设计报表。

练习 5 - 报建 WebKit 报表
创建一个报表,显示会议名称、日期、持续时间、完成百分比,负责任的名字和与会者名单

12.4 WebKit 与锐浪报表结合

Base: amos_rubylong

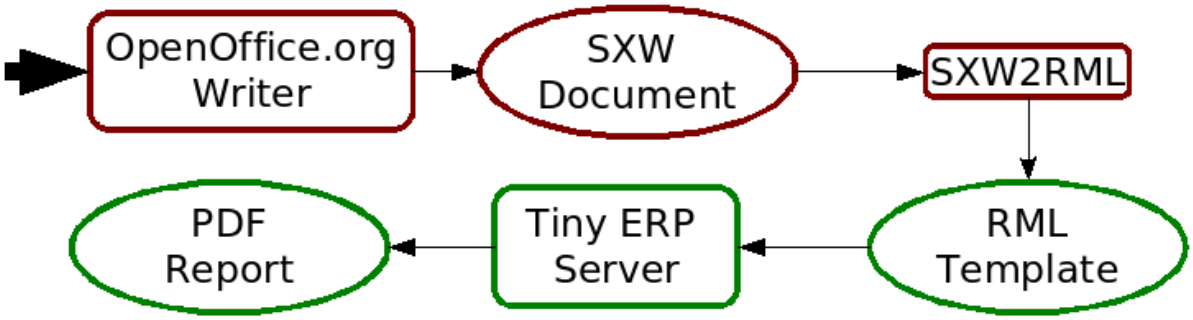
这是一个基础模块，内包含了 JS 文件与 WebKit 文件

费用报销模块：amos_rubylong_hr_expense_print

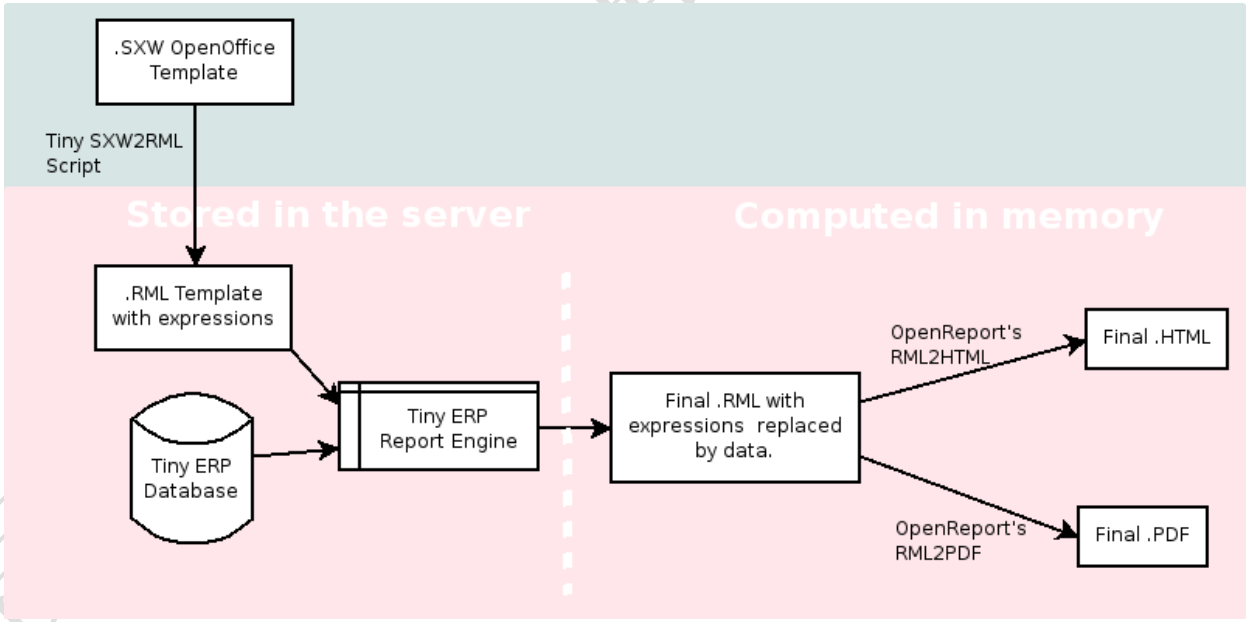
12.5 OpenOffice.org 报表

文件流

OpenOffice.org 是 Odoo 中最常用的报表格式。OpenOffice.org Writer 被用来生成 RML 模板，而 RML 模板用来生成 pdf 报表。



内部过程



.SXW 模板文件

我们用 OpenOffice1.0 的.sxw 文件作为模板。模板中包含用中括号括起来的表达式或 OpenOffice 字段(field)，用来提供给 Odoo server 填充数据。但这只是作为帮助开发者更直观地生成.RML 文件的一种方法。Odoo 并不需要 SXW 文件产生报表。

.RML 模板

可以使用 Open SXW2RML 从 SXW 文件生成 RML 文件。RML 文件是一种表现 PDF 文档的 XML 格式，可以转换为 PDF 文件。RML 是一种更容易的方法，至少 XML 语法比 PDF 语法更加通俗易懂。

.报表引擎

报表引擎(report engine)从数据库中读出数据，插入在 RML 文件的表达式中。

在.RML 文件中，将以发票上客户的城市名称替换掉相应表达式。报表引擎以真实数据替换所有表达式之后，生成相同的.RML 文件。

.生成最终文档

最后 RML 文件会根据 OpenReport 代码的需要，转换成 PDF 或者 HTML 文件。

OpenOffice 报表中的动态内容

动态内容

SXW/RML 报表中，你可以在中括号中加入 Python 代码，以获得 Odoo 中的对象(object)。代码可以使用如下变量

可用的变量

可以用的 Python 对象/变量:

objects：将要打印的 object 记录(例如发票（invoice)对象）.

data：向导(wizard)中获得的数据

time：Python 的 time 模块(详见 Python 文档).

user：运行这个报表的用户.

可以用的 Python 函数

setLang('fr')：设置语言，用于自动取得对应翻译.

repeatIn(list, varname[, tagname])：遍历模板当前部分 list 中的对象（整个文档，当前段落，表格中的当前行），可以在模板中使用 varname 作为变量名。 从 4.1.X 版开始，你可以使用第三个(可选的)参数指定循环结果放在哪个.RML 标记中.

setTag('para','xpre')：在由 sxw 转换 rml 文档过程中，替换指定标记。这里是用 xpre 替换 para (xpre 是一个预定义格式的段落)。

removeParentNode('tr')：移除类型'tr'的父结点，这个参数经常在条件语句中使用（如下例)

标签示例

[[repeatIn(objects,'o')]]：循环 objects，指定变量名为 o.

[[repeatIn(o.invoice_line,'l')]]：对 o 的 invoice_line 循环，指定变量名为'l'.

[[repeatIn(o.invoice_line,'l', 'td')]]：循环每行，并为每行数据创建一个单元格

[[(o.prop=='draft')and 'YES' or 'NO']]：根据变量‘prop’输出 YES 或 NO

[[round(o.quantity * o.price * 0.9, 2)]]：可以进行变量计算.

[['%07d' % int(o.number)]]：数字的格式化输出

[[reduce(lambda x, obj: x+obj.qty , list , 0)]]：列表中所有对象 qty 字段值的和（可以试一下用 “object” 作为列表变量）

[[user.name]]：打印报表的当前用户名

[[setLang(o.partner_id.lang)]]：从变量中取得语言

[[time.strftime('%d/%m/%Y')]]：以 dd/MM/YYYY 格式输出时间，查阅 python 文档获得关于“%d”的帮助，...

[[time.strftime(time.ctime()[0:10])]] 或 [[time.strftime(time.ctime()[-4:])]]：只输出日期.

[[time.ctime()]]：输出当前日期 & 时间

[[time.ctime().split()[3]]]：只输出时间

[[o.type in ['in_invoice', 'out_invoice'] and 'Invoice' or removeParentNode('tr')]]：如果 type 是 ‘in_invoice’ 或 ‘out_invoice’ 那么输出 ‘Invoice’；如果 不是，‘tr’ 类型的父节点会被删除.

一个有趣的标记(tag)：如果想输出当前记录的创建者(create_uid)或者最后一位修改者你需要在报表要输出的类中加入如下字段:

'create_uid': fields.many2one('res.users', 'User', readonly=1)

然后在报表中用这个变量输出相应的名字:

o.create_uid.name

有时你希望根据条件打印。 你可以用 python 的逻辑操作符 " not " , " and " , " or " 构造自己的判断语句。 Python 中的每个对象都有自己的逻辑值(TRUE 或 FALSE):

(o.prop=='draft') and 'YES' or 'NO' #prints YES or NO

注意 and 要比 or 的优先级高，表达式等价为:

((o.prop=='draft') and 'YES') or 'NO'

如果 o.prop 是 ‘draft’，那么计算结果为:

o.prop == 'draft' 为 True.

True and 'YES' 为 'YES'. 左项为 真时，返回右项，否则返回左项.这里返回字符串'YES'.

'YES' or 'NO' is 'YES'. 左项为真， or 操作会忽略右项。只计算左项值.返回字符串'YES'.

如果 o.prop 是 ‘confirm’ 之类其他的操作，那么计算如下:

o.prop == 'draft' 为 False.

False and 'YES' is False. 因为左项为 "false" 值, and 操作截断并忽略右项. 只计算左项.

False or 'NO' is 'NO'. 因为左项为 "false" 值, or 计算右项.

可以使用更复杂的的结构. 可以参照 python 手册章节 `Python's boolean operators`_ .

可以用 python 的 “filter” 函数。示例:

repeatIn(filter(lambda l: l.product_id.type=='service' ,o.invoice_line), 'line')

只输出每段中 product 含有 type=’ service’ 的行.

报表中显示二进制字段图像

[[setTag('para','image',{'width':'100.0','height':'80.0'})]] o.image or setTag('image','para')

[[p['image'] and setTag('para','image',{'width':'78.0','height':'78.0'}) or removeParentNode('blockTable')]]

[[p['image']]]

RML 报表

练习 1 - 安装 OpenOffice 的插件
安装报表设计模块，安装 OpenOffice 按装 Odoo 的插件

练习 2 - 创建报表数据。
创建一个新的表单添加每一个字段，如日期，完工百分比，负责人，会议参与人列表。添加三条

练习 3 - RML
使用 OpenOffice 来设计报告并导出 RML 文件，将其添加到您的模块，并添加一个动作

练习 4 - RML
打印报表在报表中显示所有的值，席位以红色百分比来显示

12.6 Aeroo Reports

安装文件包

1. *report_aeroo*
2. *report_aeroo_ooo*
3. *report_aeroo_printscreen*
4. *report_aeroo_sample*

打开 8100 端口

/usr/bin/soffice "--accept=socket,host=localhost,port=8100;urp;StarOffice.ServiceManager" --nologo --headless --nofirststartwizard &

第13章 WebServices

web-service 提供以下三种接口：

- SOAP
- XML-RPC
- NET-RPC

Odoo 是通过 XML-RPC 接口来操作业务，同时是一款多语言接入系统。你可以使用 Python、PHP、c#，Java 等来调用数据。

相关链接

- https://doc.Odoo.com/v6.0/developer/6_22_XML-RPC_web_services/index.html

13.1 XML-RPC

下面的例子是一个 Python 程序与 Odoo 服务器库的 xmlrpclib。

得到用户 Id

```
sock = xmlrpclib.ServerProxy('http://localhost:8069/xmlrpc/common')
UID = sock.login('terp3', 'admin', 'admin')
```

创建一个客户与他的联系人

```
import xmlrpclib

sock = xmlrpclib.ServerProxy('http://localhost:8069/xmlrpc/object')
uid = 1
pwd = 'demo'

partner = {
    'title': 'Monsieur',
    'name': '昆山一百计算机有限公司',
    'lang': 'fr',
    'active': True,
}

partner_id = sock.execute(dbname, uid, pwd, 'res.partner', 'create', partner)

user = {
    'partner_id': partner_id,
    'name': 'Amos',
    'street': 'Rue du vieux chateau, 21',
    'zip': '1457',
```

```
    'city': 'Walhain',
    'phone': '(+32)10.68.94.39',
    'fax': '(+32)10.68.94.39',
}

    sock.execute(dbname, uid, pwd, 'res.partner', 'create', user)
```

查询客户

```
args = [('vat', '=', 'ZZZZZZ')] #query clause
ids = sock.execute(dbname, uid, pwd, 'res.partner', 'search', args)
```

读取客户

```
fields = ['name', 'active', 'vat', 'ref'] #fields to read
data = sock.execute(dbname, uid, pwd, 'res.partner', 'read', ids, fields) #ids 可以是列表
```

更新客户

```
values = {'vat': 'ZZ1ZZZ'} #data to update
result = sock.execute(dbname, uid, pwd, 'res.partner', 'write', ids, values)
```

删除客户

```
# ids : list of id
result = sock.execute(dbname, uid, pwd, 'res.partner', 'unlink', ids)
```

练习 1 - XML-RPC

查询数据，

- 插入一条数据
- 更新一条数据
- 删除一条数据
- 确认一个工作流程序

13.2 Odoo 的客户端

客户端库是 Odoo 的 Python 库使用 web 服务与 Odoo 进行通迅。它是专门为那些不想写代码的使用 xml-rpc 调用数据的人。它可以处理 xml-rpc 以及 json rpc 协议和提供一系列调用方法。

你可以在(http://pypi.python.org/pypi/Odoo-client-lib)下载它。

```
$ sudo easy_install Odoo-client-lib
```

下面是 Python 调用的例子：

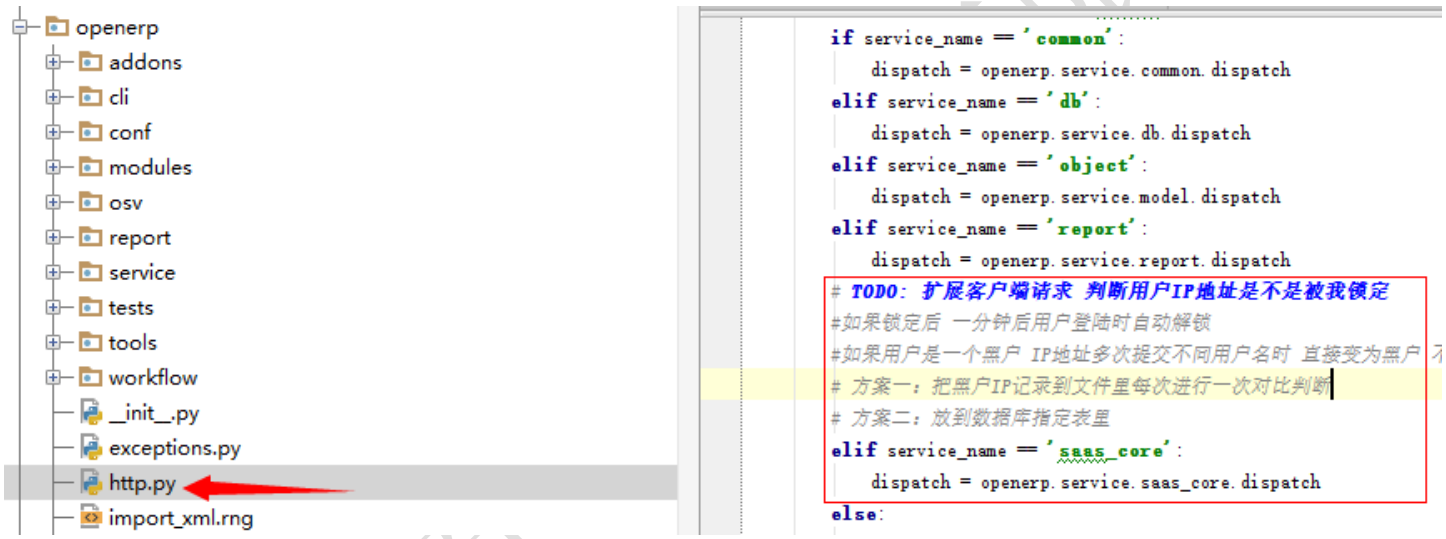
```
import Odoolib
# ... define HOST=地址, PORT=端口, DB,=数据库 USER=用户 ID, PASS=密码
  connection = Odoolib.get_connection(hostname=HOST, port=PORT, database=DB,
login=USER, password=PASS)
  connection.check_login()
  print "Logged in as %s (uid:%d)" % (connection.login, connection.user_id)

# 创建一条信息
  idea_model = connection.get_model('idea.idea') values = {
        'name':'Another idea',
        'description':'This is another idea of mine',
        'inventor_id':connection.user_id,
  }
  idea_id = idea_model.create(values)
```

练习 2 - 客户端
查询数据， ● 插入一条数据， ● 更新一条数据， ● 删除一条数据， ● 确认一个工作流程序， ● 取出报表

第14章 SAAS 设置

```
elif service_name == 'report':
    dispatch = Odoo.service.report.dispatch
# TODO: 扩展客户端请求 判断用户 IP 地址是不是被我锁定
#如果锁定后 一分钟后用户登陆时自动解锁
#如果用户是一个黑户 IP 地址多次提交不同用户名时 直接变为黑户 不许再登陆
# 方案一：把黑户 IP 记录到文件里每次进行一次对比判断
# 方案二：放到数据库指定表里
elif service_name == 'saas_core':
    dispatch = Odoo.service.saas_core.dispatch
```



Odoo 与 Python 元编程

其实 Odoo 中用到的元类（MetaClass)作用非常简单：就是在 V6.1 后我们在模块中所定义的实体类，不需要进行实例化，比如

OdooV6.1 之前的实体类定义:

```
Class MyProduct(osv.osv):
    _inherit="product.product"
    pass
MyProduct()
```

在 OdooV6.1 之后则不需要上面的类的实例化过程了。即，不需要下面这句了：

```
MyProduct()
```

为了了解元类如何实现取消实例化过程，首先我们来看一下 Odoo 中实体类的实例化过程到底做了些什么。

我们一般知道，Python 类的调用或称为实例化，会生成该类的一个实例对象(instance object),比如：

```
class A(object):
        def __init__(self, x):
```

```
        self.x = x

In [2]: a = A(2)
In [3]: a
Out[3]: <__main__.A at 0x2ff2fd0>
```

a 就是类 A 的一个实例对象。这是 Python 的基础知识，很好理解。但是，我要告诉你的是：在 Odoo 中，MyProduct()并不产生 MyProduct 类的实例，甚至再深究的话，我们经常在代码中用到的 pool.get('product.product')从对象池中获取的实例对象，也非这个 MyProduct 类所生成的实例。

这到底是怎么一回事？我们首先要了解什么是类的实例化，或者类的调用到底是怎样的一个过程，比如上例中 A(2)，其实其执行过程基本上可以分为两个部分，用 Python 来表示就是：

```
n = A.__new__(A, 2)          #创建类的实例对象
if isinstance(n, A): A.__init__(n, 2)        #实例对象初始化
```

类 A 本身并没有定义__new__类方法，所以直接调用其父类即：object 的__new__方法获得实例对象，如果获得的对象是 A 的实例则执行__init__方法

同样的，Myproduct 所继承的 osv.osv 类（或 OE6.1 以后称为 BaseModel）就有一个__new__方法，而这个方法返回的是 None，所以按照上面的说明它都不会运行到实例初始化的部分，当然也就无法获得 Myproduct 的实例。

如果我们仔细分析代码，发现这个__new__的主要作用基本上就是下面这点代码：

```
        module_model_list = MetaModel.module_to_models.setdefault(cls._module, [])
        if cls not in module_model_list:
            if not cls._custom:
                module_model_list.append(cls)
```

其实 MetaModel 是一个元类(metaclass)等会儿要讲到，module_to_models 则是这个类上的一个变量，其对应一个字典，字典的 key 对应每一个“模块”就是 Odoo 的 addons，其值对应这个模块中所定义的实体类（比如我们上例中的 MyProduct）

所以调用实体类并没有实例化，只是就这样登记备案了一下，事实上只有在模块载入(loading)过程中才会对所注册的实体类实例化，其实也不是一般意义的实例化，而是要另外创造一个新类，再做实例化。（这部分以后有空再介绍）

那么问题回到原点，OdooV6.1 以后如何做到，不调用实体类，即不运行 BaseModel 上的__new__方法就可以做到上述的类的注册过程。把 Odoo 变色的那一点黑，这就出现了。对，就是那个叫 MetaModel 的家伙。在介绍 MetaModel 之前我们先快速的讲解一下 Python 的 metaclass。

在 Python 中一切皆为“对象”， 类的实例是对象，类本身也是对象。类的实例对象是通过对类的调用获得的，那么类本身这个对象又是如何获得的呢？

其实上面的例子中类 A 的定义可以改写为：

```
A = type('A', (object,), {'__init__': lambda self,x: self.x=x})
```

从上面的代码可以看出类 A 是通过调用 type，或者是通过对 type 的实例化来获得的，事实上默认情况下所有的类都是由 type 实例

化获得，这个 type 类就是所生成类的元类。

类的实例对象可以对应五花八门我们定义的各种类，同理，我们是否可以定义除 type 以外用来生成类对象的五花八门的元类呢？答案当然是肯定的。看看我们的 MetaModel:

```
class MetaModel(type):
    ....
```

它与我们的普通的类定义没有什么差别，唯一需要注意的是其继承的父类是‘type',而在 Odoo 所有的实体类的基类 BaseModel 的类定义中有这么一句：

```
__metaclass__ = MetaModel
```

这句有特殊的含义，表示该类对象将由元类 MetaModel 实例化生成。在 Python3.x 中则用如下的语法：

```
Class MyProduct(metaclass=MetaModel, osv.osv)
```

还记得上面提到的类的实例化，__new__, __init__两步，元类的实例化也是一样。

我们看到 MetaModel 的__init__方法与上面提到的 BaseModel 类的__new__方法中有完全类似的代码：

```
        if not self._custom:
            self.module_to_models.setdefault(self._module, []).append(self)
```

就是做了一个注册备案的动作。所以类对象本身产生的过程就已经注册了类，可以不用和 6.0 及以前版本的 Odoo 每次定义实体类都要调用一下了。

第15章 开发常见问题

1.保存 openoffice 文件在英文路径下。如果不保存,send to server 的时候会没有反应

2.主要类

osv Odoo/osv/osv.py

在文件中 osv = Model

所以 osv.osv 和 osv.Model 其实是一样的

osv.Model 定义在 orm.py 中

report_sxw Odoo/report/report_sxw.py

logging python 库自带

```
import logging
_logger = logging.getLogger(__name__)
_logger.error("IntegrityError", exc_info=True)
raise osv.except_osv('xinquanda_product', "_modify_quantity 0"%(record[0]['quantity']))
```

3.使用 Odoo report design + openoffice 3.4 + Odoo 7.0

在 send to server 的时候会提示 UnicodeDecodeError: 'ascii' codec can't decode byte 通过在 addons\base_report_designer\base_report_designer.py 添加如下三行代码解决,注意默认使用空格缩进。如果增加的代码使用 tab 缩进会产生 unexpected indent 错误

```
import sys
def upload_report(self, cr, uid, report_id, file_sxw, file_type, context=None):
'''
Untested function
'''
reload(sys)
sys.setdefaultencoding('utf8')
```

4.python 一个 *.py 就是一个 package

osv.osv 就是 osv.py 文件内的 osv 对象

5.使用 parent_id 的时候，使用 toolbar 产生问题

6.selection 里面使用中文内容，需要在前面增加 u, 比如 u'供货商'。否则插入的时候会判断出错

7.使用 7.0 form 如果没有 sheet 和 group，会不显示 filed string

8.使用 Odoo report designer 自动生成 rml 会使用 in 作为 object 名。会导致在 7.0 下面无法解析。提示 cannot eval 'xxxx'之类的。修改名字解决问题

9.Win7 的字体安装直接拖进去是不行的。文件名会变成 xxx_1 xxx_2 这样(用 cmd 查看)。所以需要使用 cmd 的 xcopy 命令进行放置。否则会安装了新字体，但是还是乱码。使用 xcopy 进去以后，虽然图形界面看不到该字体安装成功了。但是重启 oe 之后可以解决乱码问题。如果还不行尝试重启一下系统吧。

10.一个工程中存在相同的 view_id 导致了显示不出来同名 menu

11.View 生成的时候调用的初始化函数

def view_init(self, cr, uid, fields_list, context=None):

使用菜单栏的导出功能，导出 Field 数据时调用

def export_data(self, cr, uid, ids, fields_to_export, context=None):

加载数据时调用，返回一个 id list.代表需要加载的数据

```
def load(self, cr, uid, fields, data, context=None):
        """
        Attempts to load the data matrix, and returns a list of ids (or
        ``False`` if there was an error and no id could be generated) and a
        list of messages.
        The ids are those of the records created and saved (in database), in
        the same order they were extracted from the file. They can be passed
        directly to :meth:`~read`
    #
    # Overload this method if you need a window title which depends on the context
    #
    def view_header_get(self, cr, user, view_id=None, view_type='form', context=None):
        return False
```

// 获取名字，返回名字列表

def name_get(self, cr, user, ids, context=None):

// 根据参数进行名字查找.返回 (id, name)的 tuple 列表.相当于先用 search 进行搜索，然后再用 name_get 获取名字列表

def name_search(self, cr, user, name='', args=None, operator='ilike', context=None, limit=100):

// 仅仅使用 name 创建 record

def name_create(self, cr, uid, name, context=None):

create

read

返回的是 dict 组成的 list

write

unlink

def search(cr, user, args, offset=0, limit=None, order=None, context=None, count=False):

cr.execute

cr.fetchall

// 定义了

def func_search(self, cr, uid, obj, name field, args, context):

12. pgsql 的备份与恢复

pg_dump.exe -f d:/backup/1234.backup -F t -h 127.0.0.1 -p 5432 -U Odoo -b Erp

pg_restore.exe -F t -h 127.0.0.1 -p 5432 -U Odoo -d tt d:/backup/1234.backup

13.

c:\>for /f "tokens=1-3 delims=- " %1 in ("%date%") do @echo %1%2%3

c:\>for /f "tokens=1-3 delims=.: " %1 in ("%time%") do @echo %1%2%3

http://www.jb51.net/article/30539.htm

14. report name 一样导致了 report 对应的 model 调用错误

15.

select 'cp' || right(cast(pow(10, 10) as varchar) || id, 10) as sn, customer_id as name, '付款' as operation, pay_value as value, date, note from xinquanda_customer_payment

union

select 'co' || right(cast(pow(10, 10) as varchar) || id, 10) as sn, customer_id as name, '退货' as operation, price_totle as value, date , '' as note from xinquanda_product_customer_out

union

select 'ci' || right(cast(pow(10, 10) as varchar) || id, 10) as sn, customer_id as name, '供货' as operation, price_totle as value, date , '' as note from xinquanda_product_customer_in;

16.有关于 view 视图的创建与显示

在.py 的对象创建里面定义_auto = False

所有_column 都需要有 readonly=True 的属性

_sql 设定视图创建 sql 语句或者在__init__(self, cr)函数里面创建具体视图,如果调用视图的 action 使用了 tree type 进行显示，可能会在报错

17. 直接在 view form 的 field 里面写 select 属性，貌似不能直接实现查询功能。需要添加 search view 来自定义搜索

a. 生成 search_view 定义

```
<record id="xinquanda_product_customer_in_search" model="ir.ui.view">
    <field name="name">xinquanda.product.customer.in.search</field>
    <field name="model">xinquanda.product.customer.in</field>
    <field name="arch" type="xml">
```

```
<search string="进货">
<field name="name" filter_domain="[('name','ilike',self),]"/>
<field name="customer_id" filter_domain="[('customer_id','ilike',self),]"/>
</search>
        </field>
      /record>
```

b. 在 action 里面指定需要的 search_view_id

```
        <record id="action_xinquanda_product_customer_in_form" model="ir.actions.act_window">
            <field name="name">客户供货</field>
            <field name="type">ir.actions.act_window</field>
            <field name="res_model">xinquanda.product.customer.in</field>
            <field name="view_type">form</field>
            <field name="view_mode">tree,form</field>
            <field name="view_id" ref="xinquanda_product_customer_in_tree"/>
<field name="search_view_id" ref="xinquanda_product_customer_in_search"/>
        </record>
```

18. domain 里面 operator 的集合

operator must be a string with a valid comparison operator from this list: =, !=, >, >=, <, <=, like, ilike, in, not in, child_of, parent_left, parent_right The semantics of most of these operators are obvious. The child_of operator will look for records who are children or grand-children of a given record, according to the semantics of this model (i.e following the relationship field named by self._parent_name, by default parent_id.[('name', 'like', 'Behave')] =>

name LIKE '%Behave%'

[('name', 'ilike', 'Behave%')] =>

name LIKE '%Behave%%'

[('name', '=like', 'Behave')] =>

name LIKE 'Behave' (no sense to use it like this)

[('name', '=ilike', 'Behave%')] =>

name LIKE 'Behave%' (with =ilike, we can control more finely the criteria)

20. 有 name_get 函数的列如果要实现查找，需要对应实现 name_search 函数

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

```
if isinstance(ids, (list, tuple)) and not len(ids):
return []
if isinstance(ids, (long, int)):
ids = [ids]
reads = self.read(cr, uid, ids, ['name','parent_id'], context=context)
res = []

for record in reads:
name = record['name']
if record['parent_id'] and record['parent_id'][1] != u'所有货物':
name = record['parent_id'][1]+'/'+name
res.append((record['id'], name))
return res

def name_search(self, cr, user, name='', args=None, operator='ilike', context=None, limit=100):
_logger = logging.getLogger(__name__)
_logger.error("name_search name=%s args=%s operator=%s context=%s"%(name, args, operator, context), exc_info=True)

if name == '':
result = super(xinquanda_product_category, self).name_search(cr, user, name, args, operator, context, limit)
_logger.error("1 name_search result=%s"%(result), exc_info=True)
return result
#去除父类名
nameIndex = name.rfind('/')
if nameIndex != -1:
name = name[nameIndex+1:]

_logger.error("2 name_search name=%s"%(name), exc_info=True)

result = super(xinquanda_product_category, self).name_search(cr, user, name, args, operator, context, limit)
_logger.error("2 name_search result=%s"%(result), exc_info=True)
return result
```

21. 统计功能的另一种实现

直接修改 search 和 read 函数。而非通过数据库试图实现。

```
class xinquanda_statistics_product_period(osv.osv):
_name = 'xinquanda.statistics.product.period'
_description = u'货物销售时间段统计'
_auto = False
_search_context = {}
def search(self, cr, uid, args, offset=0, limit=None, order=None, context=None, count=False):
_logger = logging.getLogger(__name__)
_logger.error("search args:%s, offset=%s limit=%s context:%s order=%s"%(
args,
offset,
limit,
context,
order), exc_info=True)

_search_context = {}
date_start = '2010-01-01'
date_end = '2100-12-31'

forarg in args:
if arg[0] == 'date_start':
self._search_context['date_start'] = arg[2]
if arg[0] == 'date_end':
self._search_context['date_end'] = arg[2]
if arg[0] == 'product_id':
self._search_context['id']

whereSql = ' where true'
if self._search_context.get('date_start'):
date_start = self._search_context.get('date_start')
whereSql = whereSql + " and i.date >= '%s'"%(date_start)

if self._search_context.get('date_end'):
date_end = self._search_context.get('date_end')
whereSql = whereSql + (" and i.date <= '%s'"%(date_end))
```

```
groupSql = ' group by product_id'
orderSql = ''
offsetSql = ''
limitSql = ''

if order:
orderSql = 'order by %s'%(order)

if offset > 0:
offsetSql = ' offset %d'%offset
if limit:
limitSql = ' limit %d'%limit

selectSql = """select product_id as id, product_id, '%s' as date_start, '%s' as date_end, sum(l.quantity) as quantity_totle, sum(l.quantity * l.price_sale) as price_totle, sum(l.quantity * (l.price_sale - p.price_buy)) as profits from xinquanda_product_customer_in_list as l join xinquanda_product as p on l.product_id = p.id join xinquanda_product_customer_in as i on l.in_id = i.id %s %s %s %s %s"""%(date_start, date_end, whereSql, groupSql, orderSql, offsetSql, limitSql)

_logger.error("search selectSql=%s"%(
selectSql), exc_info=True)

cr.execute(selectSql)
records = cr.fetchall()

result = []
for record in records:
result.append(record[0])

return result

def read(self, cr, user, ids, fields=None, context=None, load='_classic_read'):
_logger = logging.getLogger(__name__)
_logger.error("read ids:%s, fields:%s context:%s load=%s self._search_context=%s"%(
ids,
fields,
```

```
context,
load,
self._search_context), exc_info=True)

date_start = '2010-01-01'
date_end = '2100-12-31'

whereSql = ' where true'
if self._search_context.get('date_start'):
date_start = self._search_context.get('date_start')
whereSql = whereSql + " and i.date >= '%s'"%(date_start)

if self._search_context.get('date_end'):
date_end = self._search_context.get('date_end')
whereSql = whereSql + (" and i.date <= '%s'"%(date_end))

if isinstance(ids, (int, long)):
ids = (ids,)
if isinstance(ids, tuple):
ids = list(ids,)

if len(ids) > 0:
temp = '%s'%(ids)
temp = temp.replace('[', '(', 1)
temp = temp.replace(']', ')', 1)
whereSql = whereSql + ' and product_id in ' + temp
groupSql = ' group by product_id;'

selectSql = """select product_id as id, product_id, '%s' as date_start, '%s' as date_end, sum(l.quantity) as quantity_totle, sum(l.quantity * l.price_sale) as price_totle, sum(l.quantity * (l.price_sale - p.price_buy)) as profits from xinquanda_product_customer_in_list as l join xinquanda_product as p on l.product_id = p.id join xinquanda_product_customer_in as i on l.in_id = i.id %s %s"""%(date_start, date_end, whereSql, groupSql)

_logger.error("read selectSql=%s"%(
selectSql), exc_info=True)
```

```
cr.execute(selectSql)
result = cr.fetchall()

#把 result 转化为 record 要求的 dict 格式
colum_field = ['id', 'product_id', 'date_start', 'date_end', 'quantity_totle', 'price_totle', 'profits']
read_result = []
for record in result:
read_record = {}
index = 0
while index < len(colum_field):
read_record[colum_field[index]] = record[index]
index = index + 1
read_result.append(read_record)

#清空 search
self._search_context = {}
return read_result

_columns = {
'product_id' : fields.many2one('xinquanda.product', '货物', readonly=True, select=True),
'date_start': fields.date('起始日期', readonly=True, required=True),
'date_end': fields.date('结束日期', readonly=True, required=True),
'quantity_totle' : fields.integer('销售总量', readonly=True, select=True),
'price_totle' : fields.float('销售总额', readonly=True),
'profits' : fields.float('利润', readonly=True), }
xinquanda_statistics_product_period()#对象定义结束
```

22. 权限分配可以开启技术菜单。通过配置组的权限来实现
23. 设置表格默认列数。在 action 的 xml 里面增加<field name="limit">lines number</field>
24.设置表格默认顺序。在 py 定义里面增加 _order = 'field name [desc]'

第16章 QWEB 开发

16.1 Js 说明

```
openerp.amos_demo = function (instance) {
    instance.web.client_actions.add('example.action', 'instance.amos_demo.Action');
    instance.amos_demo.Action = instance.web.Widget.extend({

        className: 'oe_amos_demo',
        start: function () {
            this.$el.text("Hello, world!");
            return this._super();
        }
    });
};
```

amos_demo 为文件夹名

className: 'oe_amos_demo', 为 css 样式

instance.amos_demo.Action 注意 Action 为大写

16.2 CSS 样式

```
.openerp .oe_amos_demo {
    color: white;
    background-color: black;
    height: 100%;
    font-size: 400%;
}
```

.Odoo 是 OE 标准前缀

oe_amos_demo 样式名称

第17章 FAQ

17.1 Odoo 为什么选择了时区后时间还是不对？

因为你是将服务器运行在 windows 系统上的，由于 windows 系统本身的问题，Odoo 服务器实际上是无法正确取到 windows 的时区设置，所以就 fallback 到 UTC 时间。所以要使 Odoo 服务器正确设定时区，你可以在 Odoo 服务器的配置文件(Odoo_server.conf)文件中注明： timezone = Asia/Shanghai，或在 Odoo 服务器启动时，输入命令行参数：Odoo-server –timezone=Asia/Shanghai -s

或更简单的办法：如果你的服务器和客户端运行在同样的时区，不要在用户的时区里设置时区，留空即可

另一种解决方案， 在 Odoo/__init__.py 中有这样的代码

```
import os
os.environ['TZ'] = 'UTC'
import time
del os
del time
```

详见 http://docs.python.org/2/library/time.html#time.tzset

方法二：找到 dates.js 在 web\src\static\js 下第 24 行，改为

```
var obj = Date.parseExact(res[1], 'yyyy-MM-dd HH:mm:ss')
```

改完后的文件如下：

```
instance.web.str_to_datetime = function(str) {
    if(!str) {
        return str;
```

```
            }
            var regex = /^(\d\d\d\d-\d\d-\d\d \d\d:\d\d:\d\d)(?:\.\d+)?$/;
            var res = regex.exec(str);
            if ( !res ) {
                throw new Error(_.str.sprintf(_t("'%s' is not a valid datetime"), str));
            }
        *     var obj = Date.parseExact(res[1] + " UTC", 'yyyy-MM-dd HH:mm:ss zzz');
            var obj = Date.parseExact(res[1], 'yyyy-MM-dd HH:mm:ss');
            if (! obj) {
                throw new Error(_.str.sprintf(_t("'%s' is not a valid datetime"), str));
            }
            return obj;
        };
```

17.2 如何移除下拉选择列表中的“创建并编辑”链接？

github 里找到一个模块 https://github.com/0k/web_m2o_enhanced

简介翻译：

==========

此模块修改了"many2one"多对一表单字段（如此处的"订单明细 2 产品"），以便增加一些新的视图控制选项，包括：

能够让你移除 many2one 字段处下拉菜单中的"创建"和/或"创建并编辑"（译注：需要你自己去继承/修改视图，下同）；

能够让你更改 many2one 字段处下拉菜单中默认显示条目的个数；

验证权限不足时，阻止对话框的弹出。

安装此模块后，对于当前登录用户，如果没有相关对象的创建权限，默认将不显示"创建"菜单。

增加的视图控制选项:

create *boolean* (默认值: 依赖用户是否有创建权限)

控制下拉菜单中是否显示"创建"项，用户有创建权限则显示；

create_edit *boolean* (默认值: 依赖用户是否有创建权限)

控制下拉菜单中是否显示"创建并编辑"项，用户有创建并编辑权限则显示；

m2o_dialog *boolean* (默认值: 依赖用户是否有创建权限)

验证用户是否有创建权限，并决定是否显示 many2one 对话框；

limit *int* (默认值: Odoo 默认值为 7)

下拉菜单显示记录(record)个数。

举例：

```
<field name="partner_id" options="{'limit': 10, 'create': false, 'create_edit': false}"/>
```

17.3 Odoo 在哪储存附件？

http://cn.Odoo.cn/where_to_store_attachement_in_Odoo_7/

默认情况下，这些附件在 Odoo v7 中是保存在数据库中的。我们知道当附件的数量比较大时，这会严重影响数据库的性能。其实在 Odoo 中我们可以通过设置 ir.config.parameter 参数来使附件保存在文件系统中，具体菜单位置是：” 设置-技术-参

数-系统参数-ir_attachement.location” (Settings->Technical->Parameters-System parameters- ir_attachment.location）

比如我们将 ir_attachment.location 设置为 file:///filestore

那么这些附件就会保存在 Odoo 根目录/filestore 下，系统使用 sha1 哈希算法来创建文件名所以重复的文件在系统中并不会多占空间。

目前只支持 file:///协议，实际上我们可以很容易通过扩增模块来支持比如 amazons3:///协议，这样我们就可以将附件保存在亚马逊的 S3 云服务了。

数据库保存附件的模式下，数据是保存在 ir_attachment.db_datas 中

文件系统保存附件的模式下，文件名保存在 ir_attachment.db_datas_fname 中

我们尚为提供这两种模式的自动转换机制。所以，如果你设置了这个参数，那么已存在的附件仍将保存在数据库中，只有新附件会保存在文件系统中，系统会尝试访问这两个不同的位置，所以也没什么问题（先检查 db_datas,然后再检查 db_datas_fname)

注：本文末尾提供的脚本可以自动将现有数据库中的附件转换到文件系统中

如果你移除了这个参数，你需要设法将在文件系统中保存的附件存回到数据库中，因为系统就只会通过数据库来检查附件了。

将现有数据库中的附件数据转移到文件系统中的脚本(替换 URL 为您的 Odoo 实际访问 URL 地址）：

```
#!/usr/bin/python
import xmlrpclib
username = 'admin' #the user
pwd = 'password'        #the password of the user
```

```
dbname = 'database'        #the database


# Get the uid
sock_common = xmlrpclib.ServerProxy ('<URL>/xmlrpc/common')
uid = sock_common.login(dbname, username, pwd)
sock = xmlrpclib.ServerProxy('<URL>/xmlrpc/object')


def migrate_attachment(att_id):
    # 1. get data
    att = sock.execute(dbname, uid, pwd, 'ir.attachment', 'read', att_id, ['datas'])

    data = att['datas']

    # Re-Write attachment
    a = sock.execute(dbname, uid, pwd, 'ir.attachment', 'write', [att_id], {'datas': data})

# SELECT attachments:
att_ids = sock.execute(dbname, uid, pwd, 'ir.attachment', 'search', [('store_fname','=',False)])

cnt = len(att_ids)
i = 0
for id in att_ids:
```

```
    att = sock.execute(dbname, uid, pwd, 'ir.attachment', 'read', id, ['datas','parent_id'])


    migrate_attachment(id)
    print 'Migrated ID %d (attachment %d of %d)' % (id,i,cnt)
    i = i + 1


print "done ..."
```

运行这个脚本后，我们还需要清除 ir_attachements 表:

update ir_attachment set db_datas = null where store_fname is not null vacuum (full, analyze) ir_attachment

17.4 **如果调用公司名称?**

```
'complete_name': fields.function(_name_get_fnc, type="char", string='Name'),


    def name_get(self, cr, uid, ids, context=None):
        if isinstance(ids, (list, tuple)) and not len(ids):
            return []
        if isinstance(ids, (long, int)):
            ids = [ids]
        reads = self.read(cr, uid, ids, ['name','parent_id'], context=context)
        res = []
```

```
        for record in reads:
            name = record['name']
            if record['parent_id']:
                name = record['parent_id'][1]+' / '+name
            res.append((record['id'], name))
        return res

    def _name_get_fnc(self, cr, uid, ids, prop, unknow_none, context=None):
        res = self.name_get(cr, uid, ids, context=context)
        return dict(res)
```

17.5 为什么我现在在线备份不出来？

1. 可能内存不足，查看系统日志
2. 使用命令行来备份和恢复

17.6 最好使用 ORM

17.6.1 可注入的语句

```
        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 cr.fetchall()]
```

17.6.2 不可注入但是错误

```
        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 cr.fetchall()]
```

17.6.3 正确的做法

```
        auction_lots_ids = self.search(cr,uid,
                                       [('auction_id','in',ids),
                                        ('state','=','draft'),
                                        ('obj_price','>',0)])
```

17.6.4 安装完 Python 包时运行 Odoo 出错

解决方法

http://bugs.python.org/review/9291/diff/1663/Lib/mimetypes.py

```
248                 try:
249                     ctype = ctype.encode(default_encoding) # omit in 3.x!
250                 except UnicodeEncodeError:
251                     pass
252                 else:
253                     yield ctype
254                 i += 1
```

17.7 Linux 上传文件到服务器命令

1 ssh www.2cto.com

在 Cygwin 中执行：$ ssh username@remotehost

2 scp

命令 scp 基于 SSH 协议，可以将本地文件拷贝到远程服务上的指定目录，格式如下：

$ scp filename username@remotehost:remotedirectory

执行：$ scp ipmsg.log admin@10.25.1.202:/home/admin

3 ftp/sftp

首先用 root 用户登录远程 Linux 服务器，将 admin 用户添加到 FTP 账户中。

通过 echo 命令追加一行到 user_list 文件中：# echo admin >> user_list

之后通过 service 命令开启 FTP 服务：# service vsftpd start

现在就可以在本机访问 FTP 远程服务器了，然后通过 put 命令上传文件了。

在 Cygwin 中执行：$ sftp admin@10.25.1.202

4 SSH Windows Client

SSH 提供了一个 scp2.exe 作为 Windows 下的 scp 命令工具。

具体位置：C:\Program Files (x86)\SSH Communications Security\SSH Secure Shell

17.8 root 密码设置

ubuntu 的 root 密码是随机的，也就是每次开机登陆 root 的密码都是不一样的。

你现在登陆的帐户，在终端输入命令：sudo passwd 这时候系统提示：

密码：　　（输入你此时帐户的密码，系统再提示：）

输入新密码：

确认新密码：

输入新密码后就是 root 的密码，就可以用 root 用户登陆了！

17.9 Ubuntu Server+Odoo7.0 详细按装过程

Ubunte Server64 Odoo7.0-01 虚拟机提供了一个空的系统用于练习安装 Odoo

帐户：amos 密码：1

http://www.theopensourcerer.com/2012/12/how-to-install-Odoo-7-0-on-ubuntu-12-04-lts/

安装配置 ssh-server

个人比较喜欢用 SecureCRT 这样的工具来连接远程操作，所以安装 ssh-server。

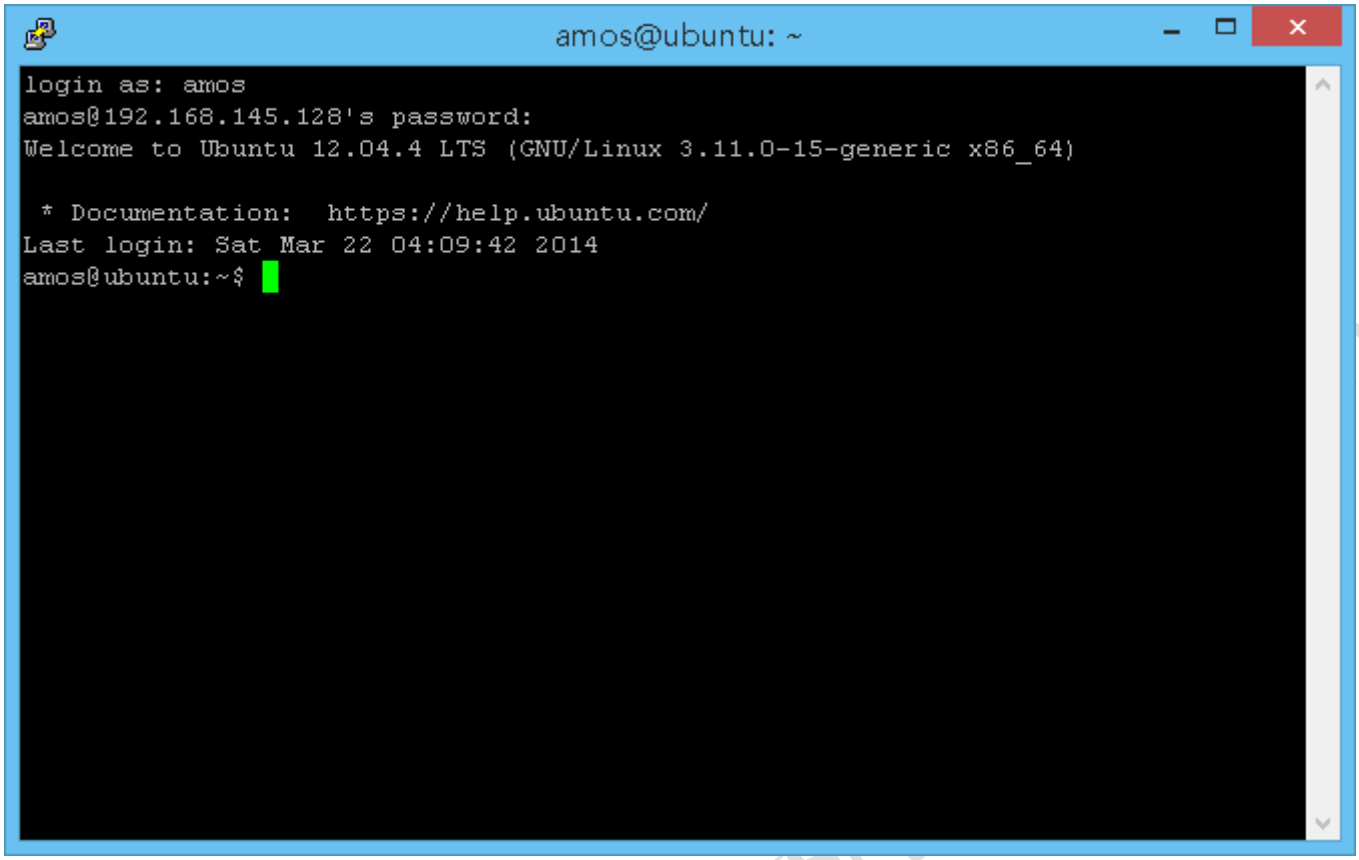
安装 ssh-server：sudo apt-get install openssh-server

启动 ssh-server：/etc/init.d/ssh restart

确认 sshserver 是否启动　：ps 空格-e 空格|grep ssh

```
amos@ubuntu:~$ ps -e |grep ssh
 1663 ?        00:00:00 sshd
amos@ubuntu:~$ _
```

安装好了以后就可以用 putty.exe 这样的工具来访问，可以直接用远程连接到刚才的 etho 口，实现远程操作维护。

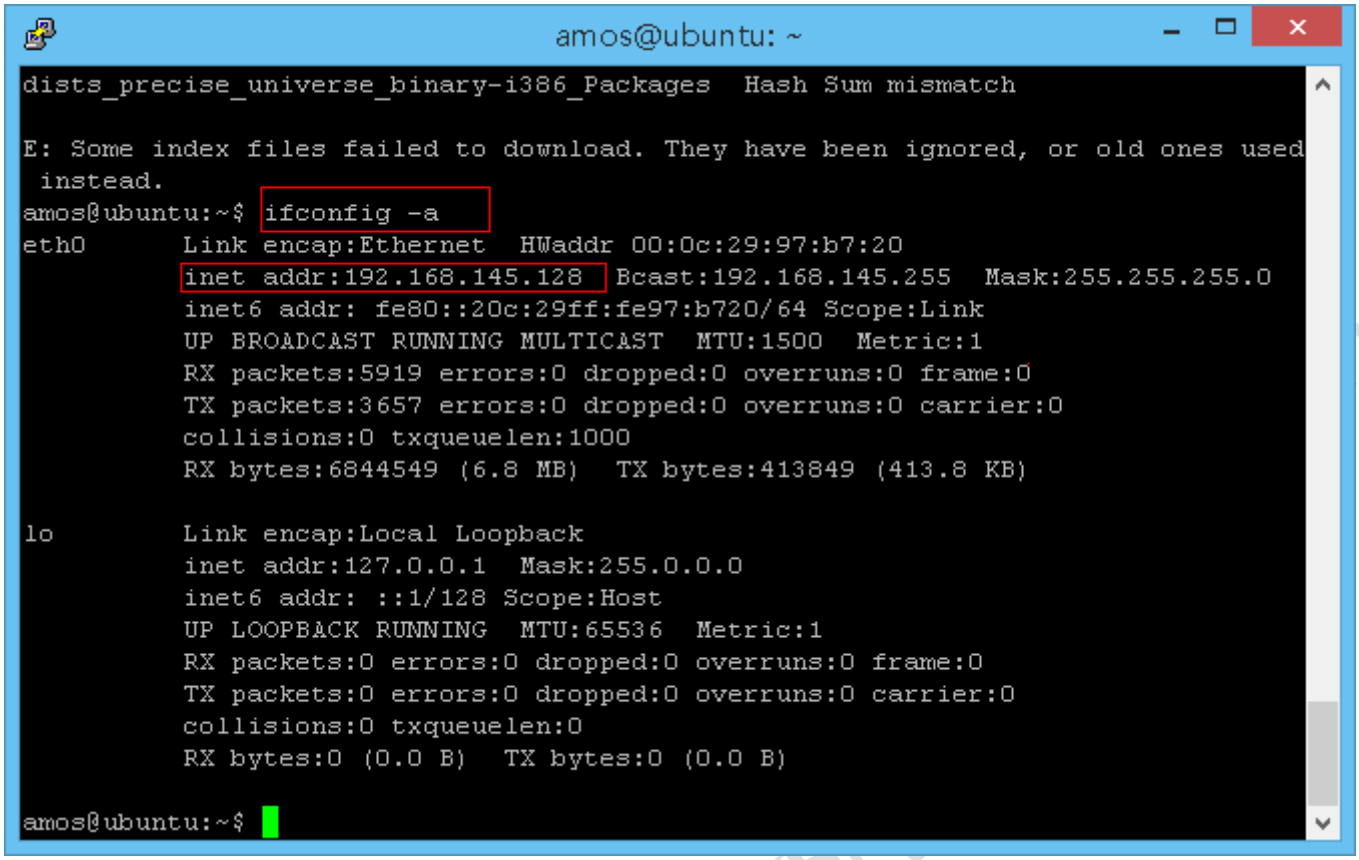


系统我们装好了，当然接下我们需要更新系统：

$ sudo apt-get update #这一步是更新你的源列表，换源后必须执行

$ sudo apt-get grade　#这一步是更新软件，如果你对新版本软件的需求不是那么迫切，可以不执行

查看本地 ip 地址：ifconfig -a



可以看到 eth0[指第一个网卡]口的 ip 地址为 192.168.145.128

安装并配置 PostgreSQL 数据库服务器

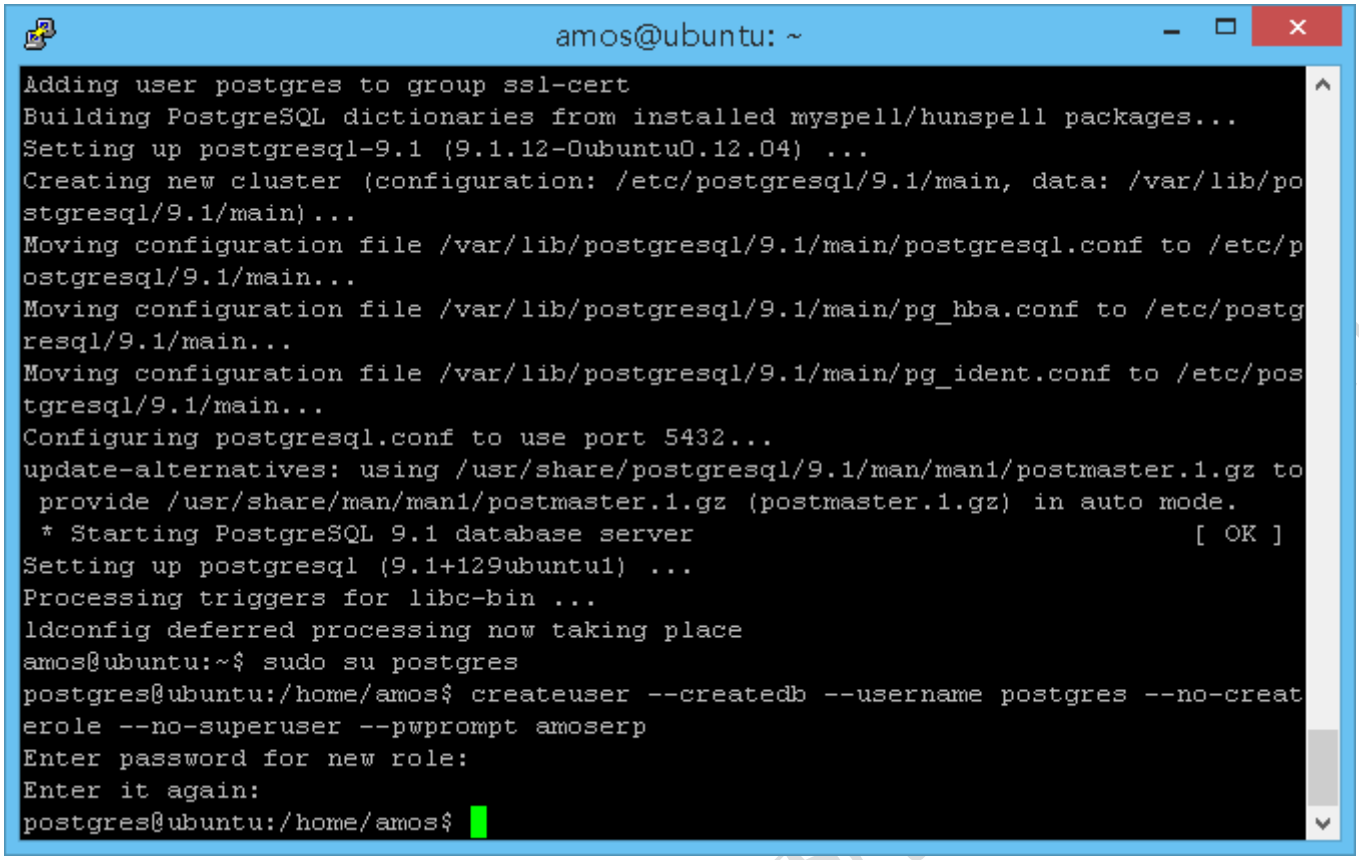
$ sudo apt-get install postgresql

PostgreSQL 默认超级用户为 ‘postgres’ . 你首次需要以此用户来登录.

$ sudo su postgres

下面是为 Odoo 在 PostgreSQL 数据库里创建一个名为 amoserp 的新用户，这里没有必要设置 amoserp 为超级用户权限:

$ createuser --createdb --username postgres --no-createrole --no-superuser --pwprompt amoserp



Enter password for new role: amoserp 这里我设置了密码为：amoserp

Enter it again: amoserp 再次输入：amoserp

这里 creatuser 命令行各选项的意思:

--createdb : 新用户能创建新数据库

--username postgres : createuser 将使用 postgres 用户 (超级用户)

--no-createrole : 此用户不允许创建新用户

--pwprompt : createuser 将询问你新用户的密码

amoserp: 新用户的名称

注意:这样 amoserp 就可以连接到 PostgreSQL 并且创建或删除数据库了，

最后 exit 退出 pgSQL：

$ exit

在 opt 下创建一个一个 Odoo7 文件夹

```
sudo adduser --system --home=/opt/Odoo7 --group Odoo7
```

安装 Odoo server

先下载 deb 安装包：

$ wget http://nightly.Odoo.com/7.0/nightly/deb/Odoo_7.0-latest-1_all.deb

然后使用 dpkg 安装 deb 包：

$ sudo dpkg -i Odoo_7.0-latest-1_all.deb

源码包进行按装:登陆 pot

wget http://nightly.Odoo.com/7.0/nightly/src/Odoo-7.0-latest.tar.gz

现在，我们需要它的代码安装：CD 到/ opt/ Odoo 的/目录并解压压缩包那里。

cd /opt/Odoo

sudo tar xvf Odoo-7.0-latest.tar.gz

修改权限给 Odoo

sudo chown -R Odoo: *

把文件夹进行改名 Odoo-7.0-20140317-003139 改成 server

sudo cp -a Odoo-7.0-20140317-003139 server

sudo apt-get install python-dateutil

sudo apt-get install python-docutils

配置的 Odoo 应用

sudo cp /opt/Odoo7/server/install/Odoo-server.conf /etc/

sudo chown amos: /etc/Odoo-server.conf

sudo chmod 640 /etc/Odoo-server.conf

sudo nano /etc/Odoo-server.conf

logfile = /var/log/Odoo7/Odoo-server.log

sudo su - amos-s /bin/bash

/opt/Odoo7/server/Odoo-server

ubuntu 下操作目录，出现 Permission denied 的解决办法

执行 Odoo7 运行命令，出现这样一个提示 Permission denied 是权限没设好

解决的办法：

$ sudo chmod -R 777 /opt/Odoo7

其中

-R 是指级联应用到目录里的所有子目录和文件

777 是所有用户都拥有最高权限

自动启动未完成

服务启动时自动运行 Odoo 服务 Odoo-server 直接复制到相关文件夹注意修改路径

sudo chmod 755 /etc/init.d/Odoo-server

sudo chown amos: /etc/init.d/Odoo-server

添加日记文件夹

```
sudo mkdir /var/log/Odoo7
sudo chown Odoo:root /var/log/opener7
```

启动服务

sudo /etc/init.d/Odoo-server start

sudo /etc/init.d/Odoo-server stop

```
sudo update-rc.d Odoo-server defaults
```

并通过 apt-get -f 选项安装依赖包：

$ sudo apt-get -f install

更新所以文件夹权限

sudo chown -R 用户名 /opt/www_bionow_com

更改文件权限

sudo chown –R 用户名 www_bionow_com.conf

Python 第三方模块中一般会自带 setup.py 文件，在 Windows 环境下，我们只需要使用命令

cd c:\Temp\foo

python setup.py install

两个命令就可以完成第三方模块的安装了。第一个 cd 命令将当前目前切换到待安装的第三方模块的

目录下（这里假设第三方模块解压后的目录为 c:\Temp\foo），第二个命令就执行安装了。安装的过程中

可能会出现“ImportError: No module named setuptools”的错误提示，这是新手很常遇见的错误提示

。不用担心，这是因为 Windows 环境下 Python 默认是没有安装 setuptools 这个模块的，这也是一个第三方

模块。下载地址为 http://pypi.python.org/pypi/setuptools。

如果是 Windows 环境的话，下载 exe 形式的安装程序就可以了（傻瓜式安装，非常快）。安装了

setuptools 之后，再运行“python setup.py install”就可以方便地安装各种第三方模块了。

如果是 Linux 环境的话，可能稍微麻烦一点，可能是笔者能力不够吧。下面简单说一下 Linux 下

setuptools 的安装过程。同样是在 http://pypi.python.org/pypi/setuptools 这个地方下载

setuptools-0.6c11-py2.7.egg 文件到本地，使用 chmod +x setuptools-0.6c11-py2.7.egg 命令使文件成

为可执行文件。然后运行 sudo sh setuptools-0.6c11-py2.7.egg 命令完成安装。

上述方法经笔者测试可行。如果有什么问题，可以留言。

17.10 安装 setuptools

官网声称.exe 版本的不支持 64 位 Windows 系统，推荐使用 ez_setup.py 自动安装。据说，有筒子 32 位的 exe 文件在 64 位系统上也能用，我就不试了，按照官方的指导来吧，免得以后出些莫名其妙的问题。


Installation Instructions

Windows

32-bit version of Python
Install setuptools using the provided .exe installer.

64-bit versions of Python
Download ez_setup.py and run it; it will download the appropriate .egg file and install it for you. (Currently, the provided .exe installer does not support 64-bit versions of Python for Windows, due to a distutils installer compatibility issue

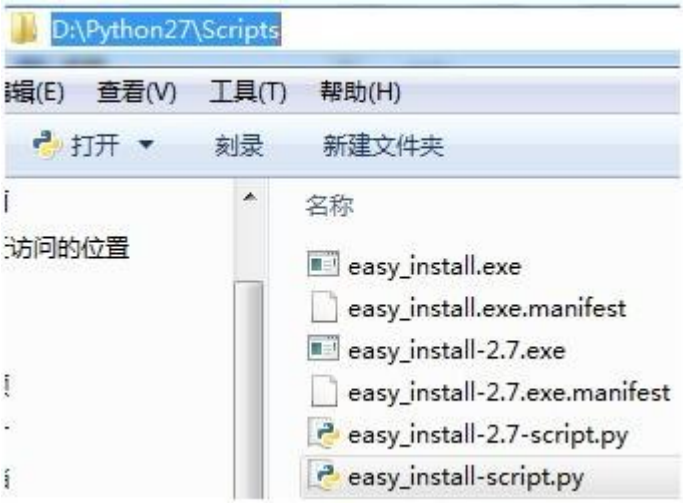
打开 http://peak.telecommunity.com/dist/ez_setup.py，把页面上的代码 copy 一份保存为 ez_setup.py。

在命令行窗口下执行即可，前提是可以联网。

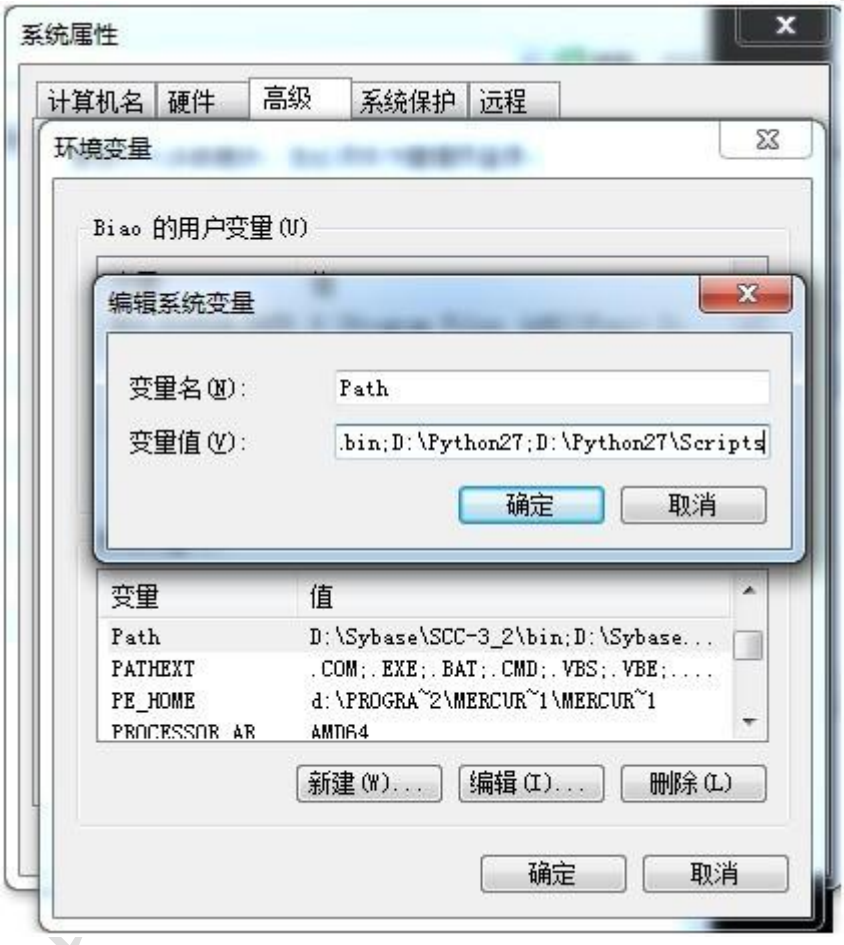
```
root@ubuntu:/home/amos# python ex_setup.py
python: can't open file 'ex_setup.py': [Errno 2] No such file or d
root@ubuntu:/home/amos# python ez_setup.py
Downloading http://pypi.python.org/packages/2.7/s/setuptools/setup
y2.7.egg
Processing setuptools-0.6c11-py2.7.egg
Copying setuptools-0.6c11-py2.7.egg to /usr/local/lib/python2.7/di
Adding setuptools 0.6c11 to easy-install.pth file
Installing easy_install script to /usr/local/bin
Installing easy_install-2.7 script to /usr/local/bin

Installed /usr/local/lib/python2.7/dist-packages/setuptools-0.6c11
Processing dependencies for setuptools==0.6c11
Finished processing dependencies for setuptools==0.6c11
```

```
1   D:\>ez_setup.py
2   Downloading http://pypi.python.org/packages/2.7/s/setuptools/setuptools-0.6c11-py2.7.egg
3   Processing setuptools-0.6c11-py2.7.egg
4   Copying setuptools-0.6c11-py2.7.egg to d:\python27\lib\site-packages
5   Adding setuptools 0.6c11 to easy-install.pth file
6   Installing easy_install-script.py script to D:\Python27\Scripts
7   Installing easy_install.exe script to D:\Python27\Scripts
8   Installing easy_install.exe.manifest script to D:\Python27\Scripts
9    Installing easy_install-2.7-script.py script to D:\Python27\Scripts
10   Installing easy_install-2.7.exe script to D:\Python27\Scripts
11   Installing easy_install-2.7.exe.manifest script to D:\Python27\Scripts
13   Installed d:\python27\lib\site-packages\setuptools-0.6c11-py2.7.egg
14   Processing dependencies for setuptools==0.6c11
15   Finished processing dependencies for setuptools==0.6c11
```



17.10.1.1 5.为 setuptools 配置环境变量



17.11 error: command 'gcc' failed with exit status 1 的解决办法

用 pip 安装软件时提示：error: command 'gcc' failed with exit status 1

easy_install 也提示一样的错误：error: Setup script exited with error: command 'gcc' failed with exit status 1

一般是缺少 python-dev 包

apt-get install python-dev 或 yum install python-devel 这样就行了

[安装包 sae-python-dev-1.3.2 安装]

第18章 ubuntu server

18.1 ubuntu 命令

18.1.1 卸载 nginx

命令：

sudo apt-get --purge autoremove nginx

命令：

which nginx

如果没有提示，证明卸载成功

18.1.2 Nginx 403 错误解决

1.检查文件目录是否有权限

解决办法 chmod 777 -R 目录名

2.未设置 index 类型

解决办法 在 ngnix.conf 中的 index 后面加上要访问的文件类型

18.1.3 Nginx 配置多个 Odoo

18.1.4 nginx 服务器重启命令，关闭

nginx -s reload ：修改配置后重新加载生效
nginx -s reopen ：重新打开日志文件
nginx -t -c /path/to/nginx.conf 测试 nginx 配置文件是否正确
关闭 nginx：
nginx -s stop :快速停止 nginx
quit ：完整有序的停止 nginx
其他的停止 nginx 方式：
ps -ef | grep nginx
kill -QUIT 主进程号 ：从容停止 Nginx
kill -TERM 主进程号 ：快速停止 Nginx
pkill -9 nginx ：强制停止 Nginx
启动 nginx:
nginx -c /path/to/nginx.conf
平滑重启 nginx：
kill -HUP 主进程号

18.1.5 查看 ubuntu 的版本的命令

sudo lsb_release –a

18.1.6 ubuntu 添加管理员权限

sudo vim /etc/sudoers

18.1.7 ubuntu 和 centos 的时间更新操作

在 Ubuntu Server 上，设置 NTP 时间同步非常简单，就如下几步：

第一，可以先进行手动更新一次时间（可选）：

sudo ntpdate ntp.ubuntu.com

第二，创建一个定时执行的文件：

sudo vim /etc/cron.daily/ntpdate

然后在其中添加一行：ntpdate ntp.ubuntu.com，保存退出。

第三，修改这个定时执行文件的权限，使其变成可执行文件：

sudo chmod 755 /etc/cron.daily/ntpdate

==========

下面解析一下,第一句是把当前时区调整为上海就是+8 区,想改其他时区也可以去看看 /usr/share/zoneinfo 目录;

然后第二句是利用 ntpdate 同步标准时间.

没有安装 ntpdate 的可以 yum 一下:

yum install -y ntpdate

加入定时计划任务，每隔 10 分钟同步一下时钟

crontab -e

0-59/10 * * * * /usr/sbin/ntpdate us.pool.ntp.org | logger -t NTP

这样，我们就可以来解决在 CentOS 系统中时间不准确的问题了。

$$$

如果执行命令出现一下错误

1.提示：7 Dec 19:24:55 ntpdate[2120]: the NTP socket is in use, exiting

这个是你 linux 机器上已经存在这个进程，输入：ps -ef | grep ntpd

Kill 掉 ntp 的进程

2.提示：No Server suitable for synchronization found

这个是最容易出现的问题，比较常见的是配置好服务器并启动服务器进程后，马上

启动客户进程，那么客户进程就会报错。解决方法是，在大约 3-5 分钟以后启动进程就行

==========

SAAS 客户端

全系统表增加关联服务 Id

Opener/models.py 510 行

add('saas_id', fields.Integer(string=u'标记', automatic=True))

add('saas_guid', fields.Char(size=40, string=u'GUID', automatic=True))

addons/web/controllers/main.py 957 行

加一个方法，用户自己扩展同步模块信息

Service/server.py 259 行

def cron_spawn(self):

""" Start the above runner function in a daemon thread.

The thread is a typical daemon thread: it will never quit and must be terminated when the main process exits - with no consequence (the processing threads it spawns are not marked daemon).

"""

try:

import amos_sync.synchronize as a

```
            tex =a.sync_thread('sync thread')
            tex.setDaemon(True)
            tex.start()
        except:
            _logger.info('Start Synchronize Faild')
            pass
```

opener/service/ 下放入

saas_client.py

saas_core.py

并在__init__.py 引入

自定义 API 协议

```
import saas_client
import saas_core
```

在 Odoo/http.py 89 行

```
threading.current_thread().uid = None
threading.current_thread().dbname = None
if service_name == 'common':
    dispatch = Odoo.service.common.dispatch
elif service_name == 'db':
    dispatch = Odoo.service.db.dispatch
elif service_name == 'object':
    dispatch = Odoo.service.model.dispatch
```

```
elif service_name == 'report':
    dispatch = Odoo.service.report.dispatch
# TODO:  扩展客户端请求  判断用户 IP 地址是不是被我锁定
#如果锁定后  一分钟后用户登陆时自动解锁
#如果用户是一个黑户  IP 地址多次提交不同用户名时  直接变为黑户  不许再登陆
#  方案一：把黑户 IP 记录到文件里每次进行一次对比判断
#  方案二：放到数据库指定表里
elif service_name == 'saas_core':
    dispatch = Odoo.service.saas_core.dispatch
else:
    dispatch = Odoo.service.wsgi_server.rpc_handlers.get(service_name)
result = dispatch(method, params
```

app.conf 文件加入

```
[options]
; admin_passwd = admin
db_host    = 127.0.0.1
db_port = 5432
db_user = oe8
db_password = 123456
db_name = s2
addons_path = addons,saas_client,saas_company,odoo
saas_client_to_service = http://127.0.0.1:8049
```

netrpc_port = 8060

xmlrpc_port = 8069

xmlrpcs_port = 8061

缓存使用

修改时

```
def write(self, cr, uid, ids, values, context=None):
        self.cache_restart(cr)
        return super(res_company, self).write(cr, uid, ids, values, context=context)
#
# This function restart the cache on the _get_company_children method
#
def cache_restart(self, cr):
        self._get_company_children.clear_cache(self)
```

对于常用的数据进行缓存

Web/controllers/main.py 973 行

def _call_kw(self, model, method, args, kwargs):

修改远程同步

18.2 Ubuntu 解压缩 zip,tar,tar.gz,tar.bz2

ZIP

zip 可能是目前使用得最多的文档压缩格式。它最大的优点就是在不同的操作系统平台，比如 Linux， Windows 以及 Mac OS，上使用。缺点就是支持的压缩率不是很高，而 tar.gz 和 tar.gz2 在压缩率方面做得非常好。

我们可以使用下列的命令压缩一个目录：

zip -r archive_name.zip directory_to_compress

下面是如果解压一个 zip 文档：

unzip archive_name.zip

TAR

Tar 是在 Linux 中使用得非常广泛的文档打包格式。它的好处就是它只消耗非常少的 CPU 以及时间去打包文件，他仅仅只是一个打包工具，并不负责压缩。下面是如何打包一个目录：

tar -cvf archive_name.tar directory_to_compress

如何解包：

tar -xvf archive_name.tar.gz

上面这个解包命令将会将文档解开在当前目录下面。当然，你也可以用这个命令来捏住解包的路径：

tar -xvf archive_name.tar -C /tmp/extract_here/

TAR.GZ

这种格式是我使用得最多的压缩格式。它在压缩时不会占用太多 CPU 的，而且可以得到一个非常理想的压缩率。使用下面这种格式去压缩一个目录：

tar -zcvf archive_name.tar.gz directory_to_compress

解压缩：

tar -zxvf archive_name.tar.gz

上面这个解包命令将会将文档解开在当前目录下面。当然，你也可以用这个命令来捏住解包的路径：

tar -zxvf archive_name.tar.gz -C /tmp/extract_here/

TAR.BZ2

这种压缩格式是我们提到的所有方式中压缩率最好的。当然，这也就意味着，它比前面的方式要占用更多的 CPU 与时间。这个就是你如何使用 tar.bz2 进行压缩。

tar -jcvf archive_name.tar.bz2 directory_to_compress

上面这个解包命令将会将文档解开在当前目录下面。当然，你也可以用这个命令来捏住解包的路径：

tar -jxvf archive_name.tar.bz2 -C /tmp/extract_here/

18.3 **安装 wkhtmltopdf**

先执行

apt-get install wkhtmltopdf

提示 Odoo 需要至少 wkhtmltopdf 0.12.0，检查安装的 wkhtmltopdf 的版本

wget http://downloads.sourceforge.net/project/wkhtmltopdf/0.12.0/wkhtmltox-linux-amd64_0.12.0-03c001d.tar.xz

解压再放在 user/bin 内

sudo cp wkhtmltox/bin/wkhtmltopdf /user/bin/

sudo chown root:root /user/bin/wkhtmltopdf

sudo chmod +x /user/bin/wkhtmltopdf

section_id 出错

addons/sale/report/invoice_report.py

```
 # _depends = {
#         'account.invoice': ['section_id'],
# }
```

第19章 WEB 界面开发

注意所有字段与对象都要小写

第20章 开发命名规范

工程格式：

所有代码都小写，如果使用中文开发，所有界面与 Python 代码都要使用中文，通过导出翻译文件可以制作其它语言包。

amos_project [文件夹]

---------- controllers [存放外部地址接口文件夹]

-------------------------- main.py [外部地址接口 py 文件]

----------data [初启化模块的数据的文件夹]

----------date.xml [初启化模块的数据]

----------date_base.xml [初启化模块底层数据]

----------demo [测试数据文件夹]

----------model_demo.xml [测试数据]

----------doc [开发特性文件夹文档]

----------index.rst [功能与注意事项说明]

----------changelog.rst [版本升级日志]

----------stage_status.rst [开发阶段]

----------edi [外部数据交换接口文件夹]

----------model.py [外部数据交换接口文件夹]

----------model_edi.xml [外部数据交换接口文件夹]

----------i18n [翻译文件夹]

---------- zh_CN.po [中文翻译包]

---------- zh_TW.po [繁体翻译包]

---------- en_US.po [英文翻译包]

----------models [继承对象修改文件夹]

----------report [报表模块文件夹]

----------model_report.xml [报表数据格式]

----------model _report_template.xml [报表引用模板]

----------security [权限设计文件夹]

----------model_security.xml [权限组与过渡规则]

----------ir.model.access.csv [对象权限]

----------static [静态文件引用文件夹]

----------description [APP 描述文件夹]

----------icon.png [APP 图标]

----------index.html [APP 模块介绍]

----------src [基础引用]

----------css [Qweb 引用样式文件夹]

----------img [Qweb 图片文件夹]

----------js [Qweb JS 文件夹]

----------xml [Qwe 界面定义文件夹]

----------templates [静态文件模板文件夹]

----------test [测试文件夹]

----------tests [测试用类文件夹]

----------views [Qweb 界面文件夹]

----------wizard [向导文件夹]

----------README.md [模块功能说明]

----------base.xml [用于存放公共的信息如顶层菜单]

----------sequence.xml [用于存放自动号规则]

----------report_qweb_view.xml [定义报表菜单]

----------model_workflow.xml [定义工作流]

总结：

原单上一个 model 对应一个 model_view.xml，但如果这个对象引用在其它的类中可以不建对应的类文件与视图文件。

Py 内容命名规则

```
class model (osv. Model):
    _name = " amos.qr.code"
```

方法取值前缀使用下划线

```
def _all_function(self, cr, uid, ids, field_name, arg, context=None):
```

按钮 button name 后台使用

```
def but_ state (self, cr, uid, ids, context=None):
```

XML ID 命名

```
<record id="view_form_model" model="ir.ui.view">

<record id="view_tree_model" model="ir.ui.view">
```

<record id="view_ search _moel" model="ir.ui.view">

<record id="view_calendar_model" model=" ir.ui.view ">

<record id="action_view_model" model="ir.actions.act_window">