

---

# 精通 Odoo

上海志鑫信息技术有限公司一万泽

2015 年 5 月 25 日



1	ERP 简介	1
1.1	Odoo 历史	1
1.2	ERP II 或商业智能化	2
1.2.1	什么是商业智能	4
2	Odoo 框架简介	7
2.1	python 模块分析	8
2.2	python2 还是 python3	8
3	Odoo 的安装和配置	9
3.1	PostgreSQL 数据库	10
3.2	Ubuntu14.04 下可能缺失的软件包	11
3.3	网页显示 node.js 方面	11
3.4	其他问题	12
3.5	通过命令行运行时的配置	12
3.5.1	-xmlrpc-port=8888	12
3.5.2	-addons-path=addons	12
3.5.3	数据库的一些配置	13
3.5.4	-save	13
3.6	将安装环境封装起来	13
3.7	文档编译	14
4	初入 Odoo	17
4.1	管理数据库	17
4.2	登录界面	18
4.3	Administrator 首选项	19
4.4	导入一个翻译	20
4.5	新的 Demo 用户	20

4.6	模块管理 . . . . .	21
4.7	修改公司信息 . . . . .	21
4.8	打开技术特性支持之后 . . . . .	22
4.9	进销存和财务系统的抽象讨论 . . . . .	22
4.9.1	以采购部门为例 . . . . .	23
4.10	安装和配置模块 . . . . .	24
5	创建自己的模块 . . . . .	27
5.1	快速生成模块骨架 . . . . .	27
5.1.1	python 模块的 <code>init</code> 文件 . . . . .	28
5.1.2	作为 Odoo 模块的说明文件 . . . . .	29
5.2	安装自定义模块 . . . . .	32
5.2.1	模块文件夹管理 . . . . .	32
5.3	一个简单的演示模块 . . . . .	33
5.3.1	<code>controllers</code> . . . . .	33
5.3.2	<code>views</code> . . . . .	33
5.3.3	<code>models</code> . . . . .	35
5.3.4	<code>security</code> . . . . .	37
5.3.5	美化网页 . . . . .	38
5.4	加分项: 通过 <code>pgadmin3</code> 来查看数据库 . . . . .	39
5.4.1	安装 . . . . .	39
5.4.2	连接服务器 . . . . .	39
5.4.3	图形化查询 . . . . .	40
6	Odoo 开发基础: 请假模块第一谈 . . . . .	43
6.1	纯理论讨论 . . . . .	43
6.2	定义模型 . . . . .	45
6.3	加入菜单 . . . . .	46
6.3.1	<code>act_window</code> 的属性 . . . . .	48
6.3.2	<code>menuitem</code> 的属性 . . . . .	48
6.4	视图优化 . . . . .	48
6.4.1	修改 <code>tree</code> 视图 . . . . .	49
6.4.2	修改 <code>form</code> 视图 . . . . .	49
6.5	完整的 <code>views.xml</code> . . . . .	51
6.6	给模块加个图标 . . . . .	53
7	Odoo 开发基础: 工作计划模块第一谈 . . . . .	55
7.1	数据访问权限管理 . . . . .	61

7.1.1	access rule	62
7.1.2	record rule	62
8	扩展现有模块—继承机制	65
8.1	给模块增加 field	65
8.2	修改已有的 field	66
8.3	重载原模型的方法	66
8.3.1	什么是 Recordset	67
8.3.2	Odoo 里面的 domain 语法	68
8.3.3	recordset 的 search 方法	69
8.4	视图 xml 文件的继承式修改	70
8.4.1	视图元素添加	71
8.4.2	原视图元素属性修改	71
8.5	多态继承	72
8.6	修改其他数据文件	73
8.6.1	删除记录	73
8.6.2	更新数据	73
8.7	委托继承	74
9	理解模型内的数据文件	75
9.1	理解外部 id	75
9.2	使用外部 id	77
9.3	导出或导入数据文件	77
9.4	快捷输入标签	78
9.5	用 field 标签设置值	78
9.5.1	eval 语法	78
9.5.2	ref 属性	79
9.5.3	One2many 和 Many2many 的 eval 赋值	79
10	Odoo 开发基础: 请假模块第二谈	81
10.1	本例涉及到的数据库表格简介	89
10.2	workflow 概念入门	89
10.2.1	定义 workflow 对象	90
10.2.2	创建节点	91
10.2.3	创建连接	91
11	Odoo 模型层详解	93
11.1	_name	93

11.2 各个表头属性 . . . . .	93
11.3 name 字段 . . . . .	94
11.4 具体模型的数据 . . . . .	94
11.5 模型间的关系 . . . . .	95
11.6  workflow . . . . .	95
12 Odoo 视图层详解 . . . . .	97
13 附录 . . . . .	99
13.1 Odoo 里老的 API . . . . .	99
13.2 PostgreSQL 数据库命令行操作 . . . . .	99
13.2.1 命令行数据库备份 . . . . .	99
13.3 反向代理 (reverse proxy) . . . . .	99
13.3.1 安装 nginx 软件 . . . . .	100
13.3.2 强制 https 连接 . . . . .	102
13.3.3 nginx 优化 . . . . .	102
13.3.4 轮询机制 . . . . .	102
13.4 跟踪项目源码初始化进程 . . . . .	102
13.4.1 base 模块 . . . . .	104
13.4.2 web 模块 . . . . .	105
13.4.3 web_kanban 模块 . . . . .	105
13.5 配置会计科目 . . . . .	105
13.5.1 配置会计科目类型 . . . . .	105
13.5.2 配置会计科目 . . . . .	106
13.6 分录 . . . . .	106
13.7 新建业务伙伴 . . . . .	106
13.7.1 新建业务伙伴标签 . . . . .	106
13.7.2 新建客户 . . . . .	106
13.8 创建新的产品 . . . . .	107
13.9 设置会计年度 . . . . .	107
13.10 向供应商下单 . . . . .	107
13.11 会计学入门 . . . . .	107
13.11.1 财务报表 . . . . .	108
13.11.2 原始凭证 . . . . .	108
13.11.3 账户 . . . . .	108
13.11.4 分类帐 . . . . .	109
13.11.5 会计科目表 . . . . .	109
13.11.6 报告期间 . . . . .	110

13.12参考资料..... 110





## 1.1 Odoo 历史

Odoo8 的前身是“Tiny ERP”，最初是由比利时的 Fabien Pinckaers 创建的。



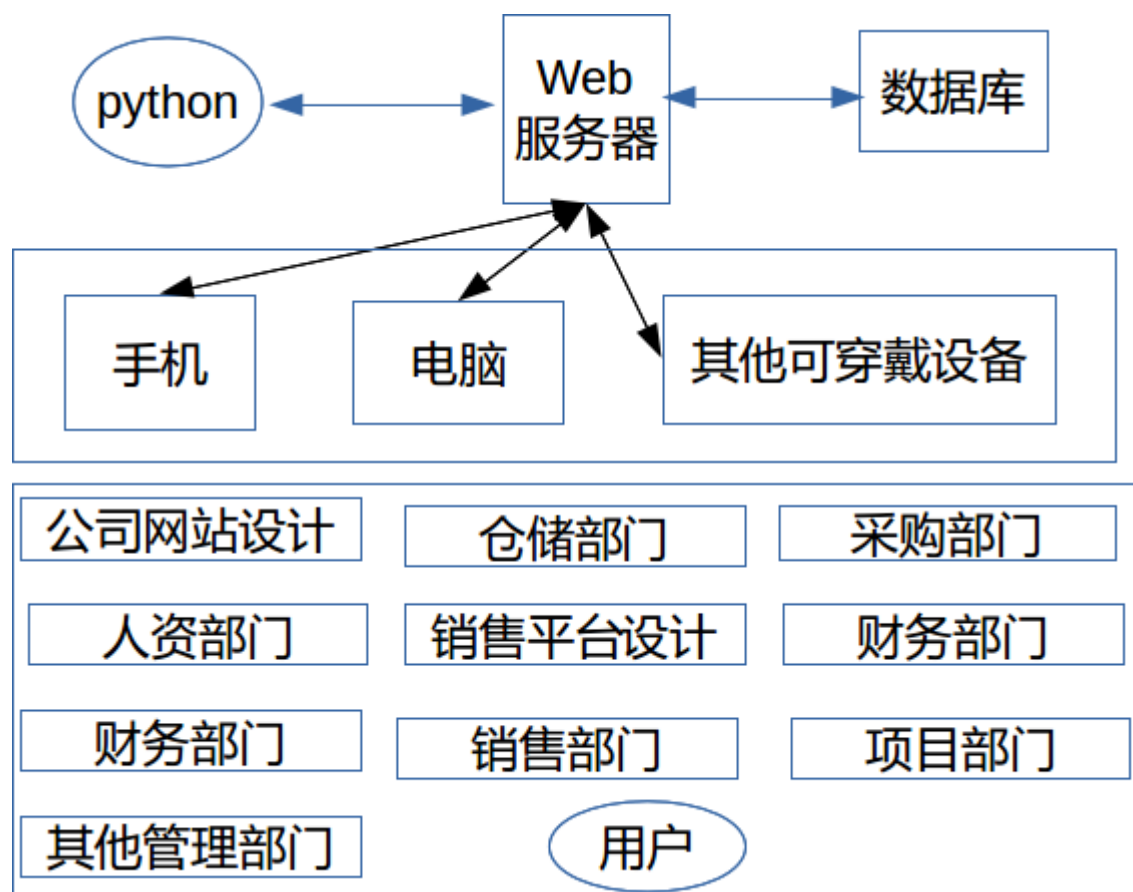
图 1.1: Fabien Pinckaers

到 2009 年的时候，发布第 5 版，公司获得风投，盈利增长迅速，软件更名为 OpenERP。OpenERP 这个名字最为人们熟知，当时软件已经包含几百个模块了，从财务管理、采

购/销售管理、库存管理到人力资源管理、销售点管理、项目管理等等都有。当时可能某些模块的功能已经开始超过传统意义上的 ERP (Enterprise Resource Planning, 企业资源规划) 的定义了 (不过最新的 ERP II 定义则更广泛, 下面会有详细的讨论。))。

而在 2014 年 9 月, 软件发布第 8 版, 在之前版本逐渐优化的 web client 这一块的基础上, 进行了大范围的功能加强。比如有了 Website builder 模块, 可以方便公司快速架构出自己的网站; e-commerce 模块方便公司快速搭建销售平台; 还有 business intelligence 这个模块, 可以辅助生成高质量的说明演示用的图形等等等等。这使得 OpenERP 这个名字已经不能很好地说明这个软件的雄心壮志了, 于是软件更名为 Odoo 这个名字了, 目前最新的版本是 Odoo8 【预计 2015 年 7 月份出 Odoo9】。

可以看得出来目前该软件的开发方向就是基于 web client/server 模型, 将公司内部所涉及到的所有的信息流都整合起来, 其不仅包括具体实施层面, 也包括分析决策层面。可以预见不久的将来 Odoo 开发将快速为公司构建出这样一个生态圈:



## 1.2 ERP II 或商业智能化

随着信息时代的到来, 商业也不可避免地走向信息化, 智能化。最新的 ERP II 的概念包含的内容如下所示:

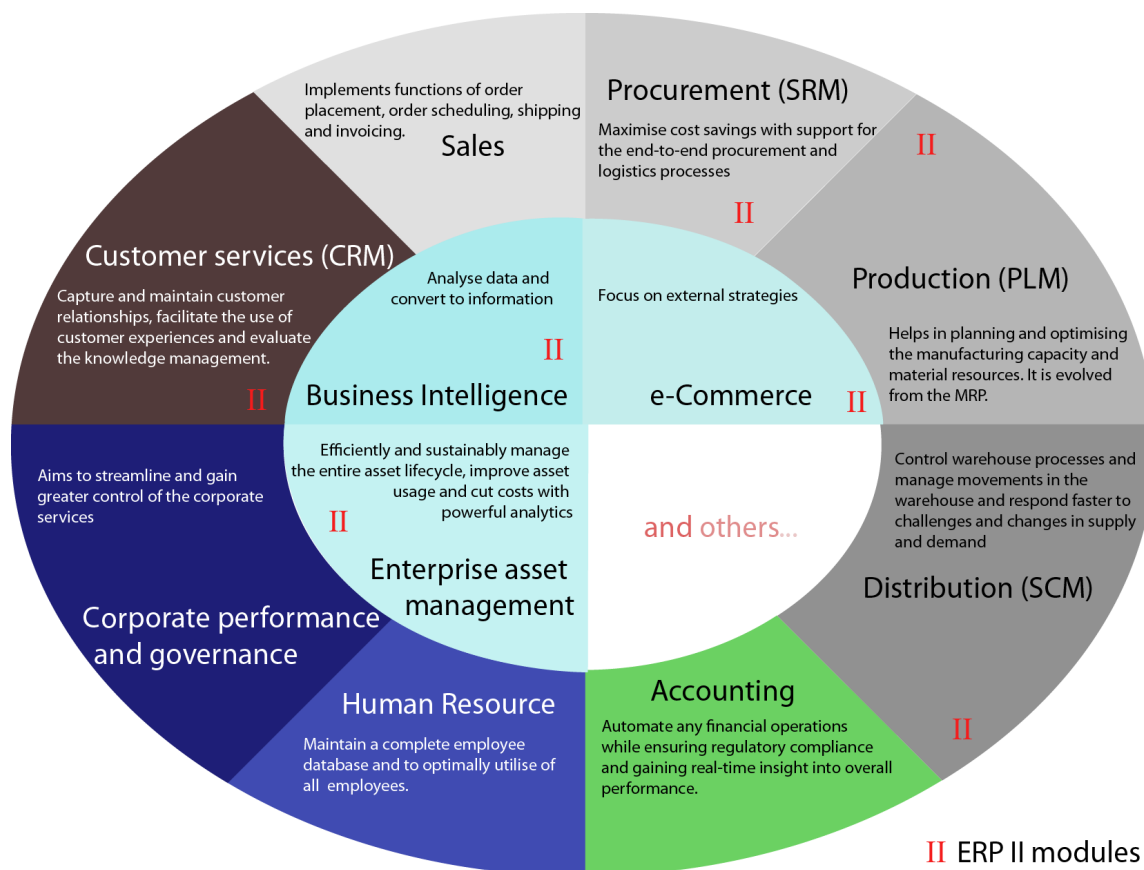


图 1.2: ERP II 模型

- **Business Intelligence** 商业智能，其主要关注于分析数据，并将数据变成知识这一过程。
- **e-Commerce** 电子商务，关注于对外战略。
- **Enterprise asset management** 企业资产管理，有效可持续地管理公司的资产生命周期，用强有力的分析工具来提高资产使用率和削减成本。
- **Procurement(SRM)** 采购，最大化的节约成本和支持终端对终端的采购，还有物流过程。
- **Production(PLM)** 生产，帮助管理和优化生产能力和物料资源。是 **MRP** 的升级版。<sup>1</sup> 这里谈论的 **PLM** 不仅要解决物料需求问题，而且要解决生产的时间问题，从而达到优化生产能力的目的。
- **Distribution(SCM)** 配送，控制仓库流程，使其能够对补给需求或更改做出快速的反应。
- **Accounting** 会计，自动化财务管理，同时要确保管理的便捷和对绩效做出实时反映。

<sup>1</sup>MRP 是 ERP 的前身，是美国生产企业为了解决物料需求问题而提出来的，主要是要解决这个问题：如果要生成多少产品，那么相应的 ABC 等等物料各自需要多少？

- **Human Resource** 人资，维护一个完整的雇员数据库，更好地使用所有雇员。
- **Corporate performance and governance** 公司表现监管，对公司的各个部门更高的控制，目标让他们能够流水线作业。
- **Customer services(CRM)** 客服，获取和维护和客户的关系，充分利用客户的体验来进行知识管理评估。（其和 **BI** 模块结合很紧密）
- **Sales** 销售，具体的定单确认，下单，货运和开发票等。

### 1.2.1 什么是商业智能

商业智能（**Business intelligence**）的概念经由 **Howard Dresner**（1989 年）的通俗化而被人们广泛了解。其将商业智能定义为：一类由数据仓库（**Data warehouses**）、查询报表、数据分析、数据挖掘、数据备份和恢复等部分组成的、以帮助企业决策为目的技术及其应用。

目前商业智能被理解为将企业中的现有数据转化为知识，帮助企业做出明智的业务经营决策的工具。这里所谈的数据包括来自企业业务系统的订单、库存、交易账目、客户和供应商资料及来自企业所处行业和竞争对手的数据，以及来自企业所处的其他外部环境中的各种数据。而商业智能能够辅助的业务经营决策既可以是作业层的，也可以是管理层和策略层的决策。

商业智能（**BI**）的架构示意图如下<sup>2</sup>。

---

<sup>2</sup>这里参考了 [这个网页](#)。

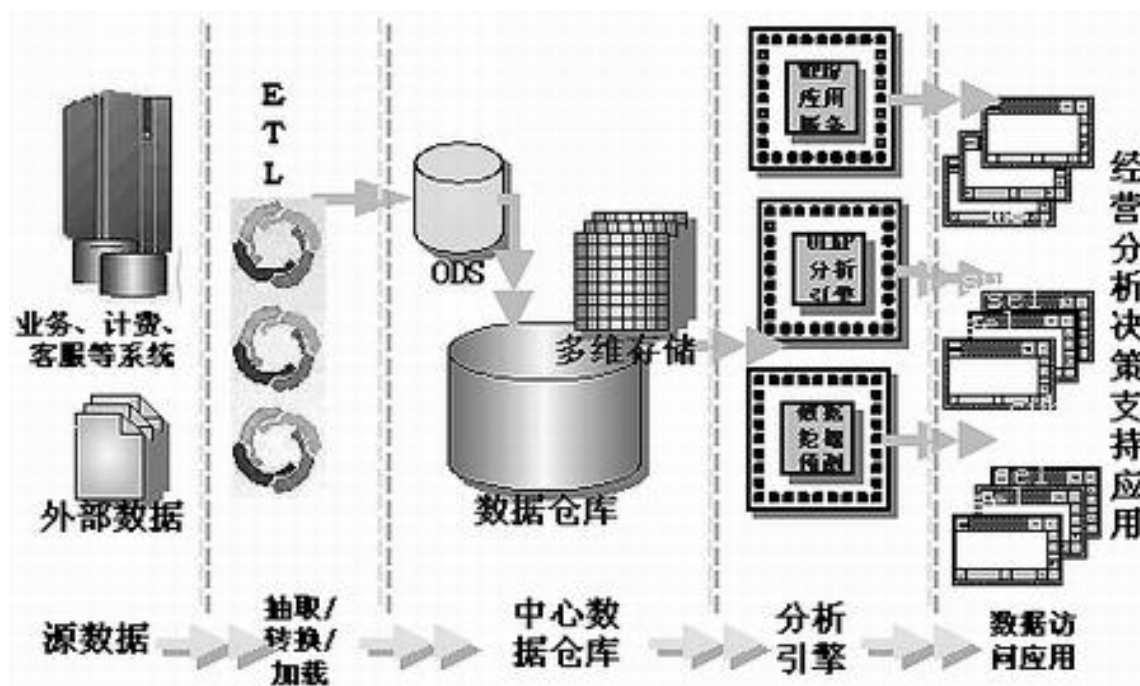


图 1.3: 商业智能技术框架

典型的 BI 系统包括：

**ETL 过程** ETL 过程是指对于数据的抽取（Extraction）、转换（Transformation）和装载（Load）。该部分从各业务系统中抽取、转换、装载数据到数据仓库。此部分通常提供一些配置手段，使得用户能够比较方便的从各种数据源取得数据，并设置规则，将数据变换成需要的形式。odoo 带有 ETL 模块。

**数据仓库** 用于存放 ETL 抽取回来的数据，此部分通常是经过数据仓库优化的关系数据库。用于数据仓库的关系数据库特别适于处理大数据量及多维数据集。odoo 的 BI 模块直接以 PostgreSQL 数据库作为数据仓库。

**OLAP** 通常是实现了 MDX（多维数据查询，Multi-Dimensional eXpress）语言的多维数据集（Cube）查询器。MDX 有些类似于 SQL，但比 SQL 更简单，是数据分析语言的事实标准。OpenERP 的 BI 模块支持 MDX 语言。

**报表工具** 用于展现 MDX 的查询结果，通常提供方便手段访问和格式化数据，提供丰富的数据呈现方式。odoo 的 BI 模块以 pyChart 作为报表工具，报表开发方法和 odoo 中的 Graph 视图类似。



## ODOO 框架简介

下面一副图很好地说明了 Odoo 技术框架:

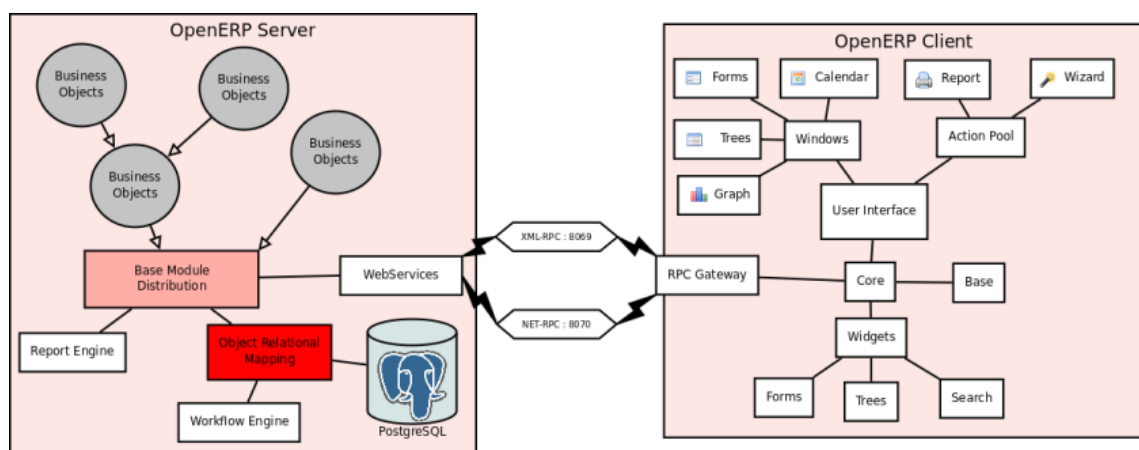


图 2.1: Odoo 技术框架

- PostgreSQL 数据库
- Object Relation Mapping 也就是大家熟知的 SQL ORM 包装层。Odoo 除了使用的基本的 `psycpg2` 作为接口之外, ORM 层是 Odoo 自己写的。
- Base Module Distribution 官方基本模块
- Report Engine 负责生成各种报表。目前支持的报表格式有 PDF, OpenOffice, HTML 三种。
- Workflow Engine 工作流引擎。支持任意复杂度的工作流。
- WebService 提供网络调用接口。目前支持 Net-RPC、XML-RPC 两种。Odoo 和

flask 一样使用 Werkzeug 作为 WSGI 层的包装, jinja2 作为模板工具。然后剩下的框架部分是 Odoo 自己写的。

## 2.1 python 模块分析

Odoo 这个框架使用了很多模块, 从这些模块的使用我们可以大致看出 Odoo 的工作原理。其中最主要的模块有:

- psycopg2 PostgreSQL 数据库接口, 其 ORM 层 Odoo 是自己写的, python 中有类似的模块 SQLAlchemy 或者 peewee 等。
- Werkzeug 和 jinja2 作为 Odoo 网络服务器框架的基础, 同样 flask 也基于这两个模块。
- babel 为网页提供国际化方案, MarkupSafe 可处理 Markdown 标记语言, lxml 用于分析网页, mock 和 unittest2 用于单元测试加强, pyserial 用于串口通信, pyusb 用于处理 usb, requests 用于处理网络协议, xlwt 用来支持 excel 表格, pillow 用于图像处理。这些模块都较好, 目前也处于活跃开发中, 已支持 python3。
- pyPDF 模块用来处理 PDF 的, 以后推荐使用 pypdf2 模块。pychart 这个模块也很快被废弃了, 关于后台运算和绘图这一块推荐使用目前流行的 ipython 系 (numpy 和 matplotlib 等) 来解决。

还有其他一些琐碎的模块十几个, 这个以后再慢慢了解。

## 2.2 python2 还是 python3

Odoo 框架严格意义上来说只支持 python2.7, 其他 python 版本都不支持, 也因为这种局限性, 也带来对其他一些 python 模块的版本号的选择性。

我做过试着编译 Odoo 的 python3 版本 (经过 2to3 脚本处理之后), 其依赖的 python 模块目前绝大部分都已经支持 python3 了, 就是 Odoo 框架自身, 要移植到 python3 问题还很大, 最大的阻碍就是他自己写 SQL ORM 那一部分。

同时我们还要考虑官方自带的模块更新也是基于 python2.7 的, 所以很长一段时间之内, 都还是安心使用 python2.7 版本的 Odoo 框架, 直到官方发布支持 python3 的版本或者我们写出一个比他更好的框架。

python2.7 和 python3 版本之间的差异在不断缩小, 但还是有很多小细节上的差异, 这块需要额外的留心。



## ODOO 的安装和配置

Odoo 项目的 **github** 地址是 <https://github.com/odoo/odoo>。我们可以看到这个项目非常活跃，下面的内容都基于 Odoo8。推荐使用 **github** 的最新版本。

常规的安装就是到 Odoo 的 **github** 地址那里下载源码，然后运行：

```
sudo python setup.py install
```

安装之，当然 Odoo 框架稍微复杂一点，这么简单处理一般是不会安装成功的，下面就一些额外的操作做出说明。

首先强调第一个问题：注意，在 **Ubuntu** 下，下载下的源码不要放在有中文名字的目录下面了，目前还不支持中文目录，包括桌面文件夹。

接下来我们 **git clone** 下 Odoo 的源码，然后 **python-dev build-essential** 是一般都要安装的，推荐先用 **apt-get** 安装上。

然后是 **pip** 工具要安装上去：

```
sudo apt-get install python-pip
```

然后运行

```
sudo pip install -U pip
```

来将 **pip** 升级到最新的工具。

`setuptools` 模块应该通过 `pip` 也安装上去了, 不太确定, 推荐用 `pip` 安装一下, 顺便也确保升级到最新的版本。

```
sudo pip install -U setuptools
```

### 3.1 PostgreSQL 数据库

PostgreSQL 是很有名的一个开源数据库, 最初由加州大学伯克利分校的计算机系开发, 其和 `sqlite3` 最大的区别就是其采用了 `client/server` 模型, Odoo 搭建在 PostgreSQL 基础之上, 也继承了这种 `client/server` 模型。Odoo 对 PostgreSQL 数据库的版本号要求不是很严格, 用最新的也是可以的。

PostgreSQL 数据库在 `ubuntu` 下安装是很方便的, 就用 `apt-get` 简单安装即可。

```
sudo apt-get install postgresql
```

但是这样安装之后, 你需要牢记一点的是, 新安装的 PostgreSQL 数据库还只有 `postgres` 这个用户有新建 `role` (或说用户) 和新建数据库的权限。你需要通过 `postgres` 这个用户来执行 `createuser` 命令才能顺利创建一个新的用户。

```
sudo -u postgres createuser $USER
```

如果某个用户不存在, 那么 PostgreSQL 将会报错:

```
createdb: could not connect to database template1: FATAL:  role "wanze" does not exist
```

你还可以通过 `postgres` 创建一个数据库:

```
sudo -u postgres createdb dbname
```

如果你的用户名已经创建了, 然后这个 `dbname` 也已经创建了, 那么你就可以用 `psql` 命令行登入这个数据库来进行相关操作了。不过 Odoo 框架要求你这个用户还具有可以创建数据库的权限。这需要你如下这样做:

```
sudo -u postgres psql postgres
ALTER USER wanze CREATEDB;
```

首先是用 `postgres` 用户身份登入 `postgres` 这个数据库, 这个数据库放着 PostgreSQL 的一些配置信息。然后使用 SQL 语句 `ALTER USER wanze CREATEDB;`

这样你的用户就有了创建数据库的权限了，这块内容参考了 [这个网页](#)。

如果系统提示你没有 `.psql_history` 这个文件，那么简单的 `touch` 一下即可。

## 3.2 Ubuntu14.04 下可能缺失的软件包

这些缺失的软件包主要和一些依赖的 `python` 模块有关。也就是如果使用 `whell` 技术封装的 `whl` 包可能是不需要安装下述的这些软件包了，不过最好还是都确保安装上吧。

关于 `python-ldap` 模块的安装参考了 [这个网页](#)，请确保下面两个软件包都安装了（否则会提示找不到 `lber.h` 错误）：

```
sudo apt-get install libldap2-dev
sudo apt-get install libsasl2-dev
```

关于 `psycopg2` 模块请确保下面软件包安装了：

```
sudo apt-get install libpq-dev
```

还有这几个软件包确保安装了，其中 `libxml2` 和 `lxml` 模块有关。

```
sudo apt-get install libxml2-dev
sudo apt-get install libxslt1-dev
```

`pillow` 模块需要安装下面这个软件包：

```
sudo apt-get install libjpeg-dev
```

随着系统的变动，你可能还缺少某些软件包，这个就要具体情况具体分析了。

## 3.3 网页显示 `node.js` 方面

按照官方文档的描述，`ubuntu14.04 nodejs` 已经安装了，只需要通过 `npm` 安装 `less` 和 `less-plugin-clean-css`，然后给 `nodejs` 创建一个别名链接 `node` 即可。

```
sudo apt-get install -y npm
sudo npm install -g less less-plugin-clean-css
sudo ln -s /usr/bin/nodejs /usr/bin/node
```

然后 ubuntu 小于 14.04 的版本还额外需要手工安装 nodejs。

### 3.4 其他问题

1. pydot 因为 googlecode 不能用了，你需要用 pip 命令单独 install pydot 将其安装好。或者到 [github](#) 这里下载之。
2. 可能有其他模块因为网络问题或者其他问题下载失败，你可以考虑用 pip 命令来安装或者到 github 下载对一个模块的源码来安装之。
3. PostgreSQL 的连接配置可能会出问题，修改 `/etc/postgresql/.../main/pg_hba.conf` 文件，看到

```
# "local" is for Unix domain socket connections only
local    all        all        peer
```

注意前面的 postgres 用户那一行绝对不要动，然后这里指本地连接，一般设置为 peer，或者干脆设置为 trust。

然后重启 PostgreSQL 服务器：

```
sudo service postgresql restart
```

### 3.5 通过命令行运行时的配置

在源码包目录下运行 odoo.py 文件即可启动 Odoo，下面是一切其他参数的配置。

#### 3.5.1 -xmlrpc-port=8888

设置网页显示的端口号，默认是 8069，但可能会出于某些原因被占用了。

#### 3.5.2 -addons-path=addons

设置插件 addons 的路径，默认会把源码 addons 文件夹加上去，但可能会出错。这里设置为源码的 addons 文件夹。

设置多个 addons 路径语法如下：--addons-path=addons, myaddons，这可以用于加载你自己定义的某些模块。

上面两个参数配置是最常用的，我就简单设置这两个参数就可以运行 Odoo 了：

```
./odoo.py --addons-path=addons --xmlrpc-port=8888
```

### 3.5.3 数据库的一些配置

这一块推荐先不设置，就使用默认的 `$USER`，然后创建数据库进入 Odoo 之后再配置。除非读者对 PostgreSQL 非常熟悉之后，再考虑更改这些配置（因为本地 `localhost` 登录还有用户权限还有连接方式上 PostgreSQL 里面还有很多内容，要小心啊。）。

- `-db_user=`

这个参数用于指定 Odoo 框架数据库的 `user`，又是不想使用默认的 `$USER`，那么可以加上这个参数。

- `-database=`

这个参数用于指定具体 Odoo 框架使用的数据库，如上你不想使用默认的 `$USER`，那么此时最好也为你想要的新用户名创建一个新的数据库。

- `-db_password=`

这个参数用于指定登录某个数据库时的密码。

### 3.5.4 -save

保存目前你的运行命令行配置，下次就可以简单使用 `./odoo.py` 来运行了。具体配置文件是主文件夹的 `.openerp_serverrc`。有兴趣的可以打开看一下。

## 3.6 将安装环境封装起来

我们在开发的时候，最好把所有依赖的 `python` 模块包版本号都固定起来，这可以通过 `python` 的 `virtualenv` 模块来实现。安装 `virtualenv` 模块是：

```
sudo apt-get install python-virtualenv
sudo pip install -U virtualenv
```

最好如上将 `virtualenv` 升级到最新的版本。

然后我们将 Odoo 框架的源码加入进去，然后再安装一次。这里不赘述了。安装成功之后，我们使用 `pip freeze` 命令来将目前的 `python` 模块包版本号环境输出出来：

```
pip freeze > requirements.txt
```

这样下次我们可以直接 `pip install -r requirements.txt` 来将依赖的 `python` 模块包刷一遍。但这还不够好。利用最新的 `wheel` 技术，我们可以将这些依赖的 `python` 模块包打包出来，从而可以实现不依赖网络的更快速的安装。

首先是安装 `wheel` 模块:

```
pip install wheel
```

然后:

```
pip wheel -r requirements.txt
```

注意将 `requirements.txt` 中的 `odoo` 版本删除，然后将 `pychart` 改成:

```
http://download.gna.org/pychart/PyChart-1.39.tar.gz#egg=PyChart
```

命令行执行完之后看到 `wheelhouse` 文件夹里面有一大堆 `whl` 文件，这就是所谓的 `pip` 模块安装包，其安装不依赖任何系统的工具，如果你有 `python` 环境，然后已经安装了 `pip` 工具，然后运行:

```
sudo pip install *.whl
```

就可以把所有依赖的 `python` 模块安装上去了，这个安装过程也不依赖于网络。

借助于 `virtualenv` 工具和 `wheel` 工具，你可以快速对你的开发环境进行再更新。比如你想升级你的开发环境的某个模块了，将其升级之后测试没有问题，就可以将其 `pip freeze` 出来，然后 `pip wheel` 出来，这样就可以快速在其他机器上部署这个开发环境了。有时可能有些模块的某些版本号 `wheel` 不出来，最好将这个模块版本号倒退。

## 3.7 文档编译

这部分是可选的，就是编译文档。需要通过 `pip` 安装 `sphinx` 和 `sphinx-patchqueue`，然后就可以 `make html` 和 `latex` 了，通过 `latex` 就可以 `make` 出 `pdf` 文档了，这方便在本机上查阅资料。

```
pip install sphinx
sudo pip install sphinx
```

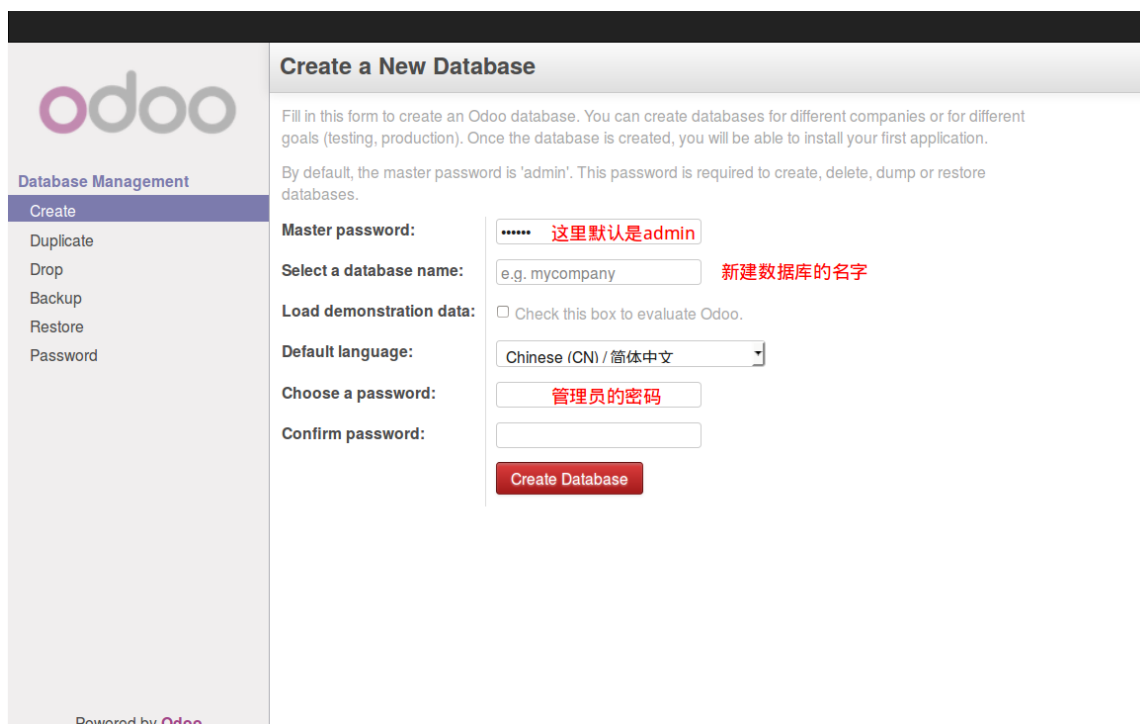
```
sudo pip install sphinx-patchqueue  
make html  
make latex
```





如上安装配置好之后，在网页浏览器上输入 **127.0.0.1:8069**（具体端口号读者要视自己的情况而定）之后我们会看到如下界面：

## 4.1 管理数据库



**odoo**

**Database Management**

- Create
- Duplicate
- Drop
- Backup
- Restore
- Password

**Create a New Database**

Fill in this form to create an Odoo database. You can create databases for different companies or for different goals (testing, production). Once the database is created, you will be able to install your first application.

By default, the master password is 'admin'. This password is required to create, delete, dump or restore databases.

**Master password:** ..... 这里默认是admin

**Select a database name:** e.g. mycompany 新建数据库的名字

**Load demonstration data:** ☐ Check this box to evaluate Odoo.

**Default language:** Chinese (CN) / 简体中文

**Choose a password:** 管理员的密码

**Confirm password:**

**Create Database**

Powered by Odoo

图 4.1：新建数据库

这里不仅新建了一个数据库，还指定了管理员的新密码，以后管理员要用那个新密码登录，用户名还是默认的 **admin**。

左边有 **create**（新建数据库），**duplicate**（复制数据库），**drop**（删除数据库），**backup**（备份数据库），**restore**（恢复数据库），**password**（修改管理员密码）。

然后这个管理数据库的 **url** 最好记下来，有的时候登入 Odoo 出问题了，这个管理数据库的界面你还是可以进入的，可以考虑还原数据库等操作。  
**http://127.0.0.1:8069/web/database/manager**。

读者可以如下用 **psql -l** 命令来查看一下，具体新建的那个数据库。

创建好数据库之后，我们就看到这个界面了，我们看到这里有很多模块。



图 4.2: 本地模块一览

## 4.2 登录界面

因为我们给 Odoo 框架创建数据库权限了，所以 Odoo 现在支持一切高级数据库操作了。我们注销 **Administor** 账户，然后就会看到一个登录界面。



图 4.3: 登录界面

然后下面就有一个管理数据库的链接，点击就进入之前看到的管理数据库界面了。

### 4.3 Administrator 首选项

在右上角 Administrator→ 首选项那里可以设置网站的语言，时区，还有管理员的头像，管理员的邮箱和个性签名。

更改我的首选项

Administrator

更改密码

语言

Chinese (CN) / 简体中文

时区

Hongkong

电子邮件选项

签名

a358003542@163.com

B I U abc T [List Icons]

--  
wanze

图 4.4: Administrator 首选项

## 4.4 导入一个翻译

看到左侧翻译一栏，点击导入一个翻译即可加载一个翻译。

## 4.5 新的 Demo 用户

看到左侧用户一栏，点击创建一个新用户，即进入创建一个新用户的界面。这里将新建一个演示用的 **Demo** 用户。



图 4.5: 新的 Demo 用户

然后我们使用这里设置的 **Demo** 用户名和密码登录，会看到一片空白，没有左侧的那些设置选项了。

## 4.6 模块管理

- 安装模块，刚进入 Odoo 即看到那个模块一览的界面，点击相应的模块即安装对应的模块了。
- 更新模块，然后看到左侧还有一个更新选项可用于更新本地模块。
- 卸载模块，按照官方文档的描述，虽然点击对应模块的详细表单视图，里面有卸载按钮，但并不推荐。最好还是每次有模块更动之前先备份一下数据库，之后不行恢复一下数据库即可。

## 4.7 修改公司信息

在最左上角那里，鼠标划过会看到编辑公司数据的信息，然后点击进入即可以看到如下的修改公司信息界面：

公司

×

志鑫

公司名称

上海志鑫信息技术有限公司

一般信息

设置

Report Configuration

地址

街道...

电话

传真

电子邮件

info@yourcompany.com

税号

公司登记

城市

省

邮编

国家

Your Company Tagline

http://www.yourcompany.com

银行帐户

银行账号	银行名称	显示报表	帐号所有者
添加一个项目			

图 4.6: 修改公司信息

比如设置公司的 **Logo**，名字，地址，网站等等。然后在设置选单那里还可以设置币种，这个币种设置会影响后面会计模块的默认币种行为。还有一些信息设置能够填上的最好都填上。

4.8 打开技术特性支持之后

打开技术特性支持之后会多了很多参数设置，比如  **workflow配置**那一栏，之前是没有的。

再比如你可以更新模块列表了，只有你更新模块列表之后，你新写的模块才能被搜索到和安装上去。

4.9 进销存和财务系统的抽象讨论

进销存和财务软件系统目前大多融为一体了，以前还是分开的。然后进销存那块以前最开始的软件是仓库管理软件，后面采购和销售是慢慢加上去的。理解这点很重要，我们可以把仓库管理看作最底层的模块，而采购和销售是于之上的模块，然后采购和销售又和财务存在着很多信息交流。具体如下图所示：

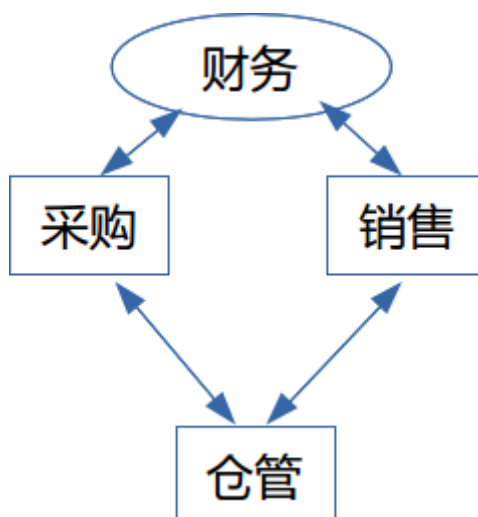


图 4.7: 进销存和财务

简单来说就是不管是采购还是销售其一笔成交的订单都必然产生两个信息流，一个是仓管那边；一个是财务那边。而这个信息流的过程最好是由系统自动化完成。下面以采购部门为例子来说明系统内部的这种信息流自动化过程。

#### 4.9.1 以采购部门为例

采购部门一般职能是接受其他各部门的采购要求，定期汇总做成采购计划；然后根据采购计划会相应的供应商询价、议价；然后下采购单；然后跟踪供货商及时发货；货到后验货、入库。如果有问题则要求供应商换货、退货。

首先看采购计划那边，本公司各个部门根据实际需要应该都可以向采购部门发送采购需求，然后某些部门的模块会根据自己的实际情况有自动发送采购需求的功能，比如仓管的最小库存原则，再比如销售部门的某些紧急需求等等（原则上其他部门不紧急的需求应该发送给仓管，然后由仓管根据最小库存原则来发送采购需求，但某些部门具有特别紧急的需求是可以直接给采购部门发送采购需求的。当然还可能某些公司的某些部门的情况是种类繁多库存较小的情况，那么也是可以直接向采购部门发送需求的）。这个信息采购需求的发送机制需要根据不同的公司实际情况进行优化。

对于采购需求，系统应该具备一定的自动整理功能，比如按照供应商分类，紧急程度插队机制等等。采购人员处理经过初步整理出来的采购需求，将这些信息自动生成 **询价单**，此时系统可以提供电邮，打印询价单的功能，或者和供应商洽谈视频的功能等。询价结束之后对方同意之后采购员可以点击确定然后将这些信息发送给采购部门的经理，由经理确认之后，**采购单**就自动生成了。



图 4.8: 这是采购单

采购单确认之后，前面讲的信息流分成仓管和财务两边。首先我们看仓管那边，在货物还没送到之前，仓管那边就可以看到将要到的货物了，这样他们可以预先对仓库进行整理，方便后面的快速存放工作。货物送到之后首先是采购员验收，验收确认采购单上单击 **收货**，然后是点击 **入库** 操作，就表示这个货物正式入库了，然后仓管那边对于是什么产品，单价多少，重量多少甚至体积多少都可能有所说明自动存放在软件系统仓管那一栏了。

然后是财务那边，在采购单确定之后，财务那边应该 **接受发票**，然后点击 **确认生效**，这是相应的信息应该送到财务那边去了。财务那边还有一些付款事宜。财务付款完了采购员就会看到这个采购单已经完成了。

这里谈论的以采购部门为例的一般流程，这个流程不是死的，也不一定是最优的。具体要根据公司的实际情况的不同和功能需求不同对这些底层的信息流程做出调整和优化。

## 4.10 安装和配置模块

我们现在就安装进销存加财务这四个经典模块：

- 财务与会计 ( **account** 模块),
- 仓库管理 ( **stock** 模块),
- 采购管理 ( **purchase** 模块),
- 销售管理 ( **sale** 模块).



安装完这些模块之后需要再导入一次翻译（否则可能会有某些词条没有正确翻译），然后重启 Odoo 框架好让这些新的模块的翻译生效。

在安装过程中会弹出这个会计窗口，默认值就可以了。



## 创建自己的模块

Odoo 开发的一条黄金准则就是我们最好不要修改现有的模块，特别是官方内置的那些模块。这样做会让原始模块代码和我们的修改混为一谈，使得很难对软件进行升级管理。我们应该创建新的模块（在原有的模块基础之上）来达到修改和扩展功能的目的。Odoo 提供了一种继承机制，允许第三方模块扩展现有模块，或者官方的或者来自社区的，这种继承修改可从任意层次来开展，从数据模型到业务逻辑到用户界面。

### 5.1 快速生成模块骨架

可以如下快速生成一个模块骨架：

```
./odoo.py scaffold mymodule myaddons
```

这将在当前位置新建一个 **myaddons** 文件夹，然后在 **myaddons** 文件夹下创建一个名字叫 **mymodule** 的模块骨架。创建好了之后读者可以进入翻一下。值得一提的是，这个模块骨架并不怎么实用，在学完本章稍微对 Odoo 的模块结构有所熟悉之后，自己新建文件或者复制一些代码过来可能反而会更加灵活便捷一些。其中有些原因是这个命令生成的模块结构有点老旧了，官方文档的推荐结构如下所示：

```
addons/<my_module_name>/
|-- __init__.py
|-- __openerp__.py
|-- controllers/
|   |-- __init__.py
```

```
|   '-- main.py
|-- data/
|   |-- <main_model>_data.xml
|   '-- <inherited_main_model>_demo.xml
|-- models/
|   |-- __init__.py
|   |-- <main_model>.py
|   '-- <inherited_main_model>.py
|-- security/
|   |-- ir.model.access.csv
|   '-- <main_model>_security.xml
|-- static/
|   |-- img/
|   |-- lib/
|   '-- src/
|       |-- js/
|       |-- css/
|       |-- less/
|       '-- xml/
'-- views/
    |-- <main_model>_templates.xml
    |-- <main_model>_views.xml
```

这里的文件夹权限推荐是 755，文件权限推荐是 644。

上面只是一个泛泛而论的情况，具体有些文件夹或文件可能是不需要的。下面对这些内容进一步说明之。

### 5.1.1 python 模块的 init 文件

Odoo 模板本质上就是一个 python 模块，所以首先它应该有一个 `__init__.py` 文件。刚开始里面不放任何内容都是可以的。

而 scaffold 自动创建有下面的内容：

```
import controllers
import models
```

推荐改成这样的形式（有更好的 `python2` 和 `python3` 兼容性）：

```
from .models import main_model
```

这种相对路径语法 `python2` 和 `python3` 都是通用的，具体对应的就是本目录下的 `models` 目录下的 `main_model.py` 文件。推荐将所有的模型定义 `python` 文件都放入 `models` 文件夹中，`models` 文件夹下也应该有一个 `__init__.py` 文件。然后这里还新建了一个 `main_model.py` 文件，这个文件的名字倒是随意的了。

### 5.1.2 作为 Odoo 模块的说明文件

然后本模块作为 Odoo 框架的模块还必须新建一个 `__openerp__.py` 文件，最小型的什么都不做的 Odoo 模块就需要这两个文件，一个是这个 `__openerp__.py` 文件，一个是 `__init__.py` 文件。

scaffold 自动创建的 `__openerp__.py` 文件大致内容如下：

```
# -*- coding: utf-8 -*-
{
    'name': "mymodule",

    'summary': """
        我的第一个模块
        """,

    'description': """
        我的第一个模块，用于学习自定义模块。
        """,

    'author': "wanze",
    'website': "http://www.yourcompany.com",

    # Categories can be used to filter modules in modules listing
    # Check https://github.com/odoo/odoo/blob/master/openerp/addons/base/module/module_data.xml
    # for the full list
    'category': 'Test',
    'version': '0.1',
```

```
# any module necessary for this one to work correctly
'depends': ['website'],

# always loaded
'data': [
    'security/ir.model.access.csv',
    'views/mymodule_templates.xml',

    'demo.xml',
],
# only loaded in demonstration mode
'demo': [
    'demo.xml',
],
}
```

以后我们要创建一个新的模块，这个文件的格式可以复制粘贴过去。其中一些基本的比如 **name** 就是本模块的名字，然后 **description** 还有 **category** 分类等等就不多说了，这些含义都是很明显的，就是本模块的一些描述信息。然后提得一提的是这三个属性：

**depends** 本模块的依赖关系，这里主要是指本模块对于 Odoo 框架内其他的模块的依赖。如果本模块实在没什么依赖，就把 **base** 模块填上去。

**data** 本模块要加载的数据文件，别看是数据文件，似乎不怎么重要，其实 Odoo 里面视图，动作，工作流，模型具体对象等等几乎大部分内容都是通过数据文件定义的。具体这些 **xml** 或 **csv** 文件如何放置后面再讲。

**demo** 这里定义的数据文件正常情况下不会加载，只有在 **demonstration** 模式下才会加载，具体就是你新建某个数据库是勾选上了加载演示数据那个选项。如下图所示：

**志鑫**

**Database Management**

- Create
- Duplicate
- Drop
- Backup
- Restore
- Password

### Create a New Database

Fill in this form to create an Odoo database. You can create databases for different goals (testing, production). Once the database is created, you will be able to in

By default, the master password is 'admin'. This password is required to create databases.

**Master password:** .....

**Select a database name:** odoo\_demo

**Load demonstration data:** ☒ Check this box to evaluate Odoo. 这里勾上

**Default language:** Chinese (CN) / 简体中文

**Choose a password:** .....

**Confirm password:** .....

**Create Database**

图 5.1: 加载演示数据

这可能并不是你想要的效果，因为其他官方内置模块也附加很多演示信息进来了。其实读者一定也想到了，我们完全可以将 `demo.xml` 放入“data”那里，然后实际运作的时候不加载就是了，我更喜欢这种处理方案。

#### 属性值清单：

**name** 模块名字

**summary** 可以看作简短介绍吧

**description** 可以看作详细介绍

**author** 模块作者

**website** 模块网站

**category** 模块分类

**version** 模块版本号

**license** 模块版权信息，默认是 AGPL-3

**depends** 模块依赖

**data** 模块必加载的数据文件

`demo demonstration` 下才加载的数据文件

`installable` 默认 `True`，可设为 `False` 禁用该模块

`auto_install` 默认 `False`，如果设为 `True`，则根据其依赖模块，如果依赖模块都安装了，那么这个模块将自动安装，这种模块通常作为胶合 (`glue`) 模块。

`application` 默认 `False`，如果设为 `True`，则这个模块成为一个应用了。你的主要模块建议设置为 `True`，这样进入 `Odoo` 后点击本地模块，然后默认搜索过滤就是 应用，这样你的主模块会显示出来。

## 5.2 安装自定义模块

就设置好这样两个文件，虽然里面什么内容都没有，实际上也就可以开始安装这个模块而来。

前面说了设置 `--addons-path=addons, myaddons`，就可以加载自定义的模块了。具体安装就和安装其他模块没有两样，除了你需要清除搜索栏然后输入搜索关键词。然后注意如果不是新建的数据库，那么需要打开技术特性执行，[更新模块列表](#) 之后，才能搜索到你自己定义的新的模块。

自定义模块如果修改之后，（不需要重新编译 `Odoo`），肯定是需要重启 `Odoo` 的，然后进去之后如果你的模块增减了额外的文件，则还需要升级 (`update`) 相应的模块。

### 5.2.1 模块文件夹管理

- `data` 文件夹，放着 `demo` 和 `data.xml`
- `models` 文件夹，放着模型定义
- `controllers` 文件夹，`http` 路径控制
- `views` 文件夹，网页视图和模板
- `static` 文件夹，网页的一些资源，里面还有子文件夹：`css`，`js`，`img`，`lib` 等等。



## 5.3 一个简单的演示模块

### 5.3.1 controllers

现在我们在 `controllers` 文件夹里新建一个 `__init__.py`，然后新建一个 `main.py` 文件。在 `main.py` 文件中添加如下内容：

```
class Mymodule(http.Controller):
    @http.route('/mymodule/mymodule/', auth='public')
    def index(self, **kw):
        return "Hello, world"
```

现在请读者如下所示安装和更新自定义模块之后，进入 `127.0.0.1:8089/mymodule/mymodule` 来看一下效果。这样我们就明白了这里的 `@http.route` 装饰器有根据具体某个函数的返回信息进行路径分发和控制访问权限的功能。

如果没有什么问题，你应该能看到一行 `hello world` 文字，祝你好运。

### 5.3.2 views

`views` 文件夹用于视图控制，里面通常放着一些模板文件等。我们首先修改 `__openerp__.py` 文件中 `data` 属性为这个样子。

```
'data': [
    'views/mymodule_templates.xml',
],
```

然后按照这个在 `views` 文件夹里面创建一个 `mymodule_templates.xml` 文件。

```
<openerp>
  <data>
    <template id="index">
      <title>MyModule</title>
      <t t-foreach="fruits" t-as="fruit">
        <p><t t-esc="fruit"/></p>
      </t>
    </template>
```

```
</data>
</openerp>
```

这里使用了 Qweb 模板语言，就这里提及的我们可以简单了解下：

```
<t t-foreach="[1, 2, 3]" t-as="i">
<p><t t-esc="i"/></p>
</t>
```

其输出是：

```
<p>1</p>
<p>2</p>
<p>3</p>
```

如果对应 python 语言的话，可以理解为：

```
for i in [1, 2, 3]:
    print('<p>{i}</p>'.format(i = i))
```

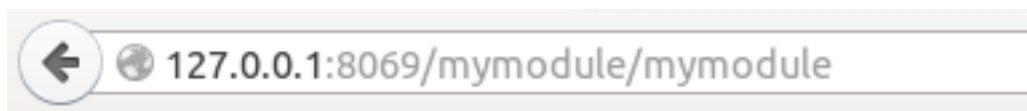
这里的 `<t t-esc="i"/>` 是先计算 `i` 的值，然后将其打印出来。

然后在之前 `controllers` 那里的 `main.py` 文件那里，我们使用这个模板文件。

```
from openerp import http

class Mymodule(http.Controller):
    @http.route('/mymodule/mymodule/', auth='public')
    def index(self, **kw):
        return http.request.render("mymodule.index",
                                   {'fruits': ['apple', 'banana', 'pear']})
```

如果不出意外的话，你应该看到这样的画面了：



apple

banana

pear

这里调用 `http.request.render` 函数，可以猜到这是一个网页模板渲染输出函数。然后注意看第一个参数，`mymodule.index`，这里还是有一些讲究的，`mymodule` 就是你正在编写的这个模块的名字，然后这个 `index` 对应的就是那个网页模板文件的 `id` 属性：`<template id="index">`。接下来的第二个参数就是给模板传递进去的一些变量值了。

### 5.3.3 models

一般一些数据对象或类就定义在 `models.py` 文件里，如果类继承自 `models.Model` 类，那么这个类就有了 Odoo 字节写的 ORM 接口了。也就是你定义的这些模型生成的对象都会存入 SQL 数据库中。

我们在 `models` 文件夹里面新加入一个 `__init__.py` 文件，然后再新建一个 `main_module.py` 文件，在其内写上一个简单的模型定义，具体如下所示：

```
class Fruits(models.Model):
    _name = 'mymodule.fruits'

    name = fields.Char()
```

然后 `__init__.py` 那边也注意加载好模块，这里就不多说了。

新加入一个演示数据 `demo.xml`：

```
<openerp>
```

```

<data>
  <record id="apple" model="mymodule.fruits">
    <field name="name">apple</field>
  </record>
  <record id="banana" model="mymodule.fruits">
    <field name="name">banana</field>
  </record>
  <record id="pear" model="mymodule.fruits">
    <field name="name">pear</field>
  </record>
</data>
</openerp>

```

在 `__openerp__.py` 的 `demo` 那里设置成这样:

```

'data': [
    'security/ir.model.access.csv',
    'views/mymodule_templates.xml',

    'demo.xml',
],

```

然后 `controllers/main.py` 改成这样了:

```

from openerp import http

class Mymodule(http.Controller):
    @http.route('/mymodule/mymodule/', auth='public')
    def index(self, **kw):
        fruits = http.request.env['mymodule.fruits']

        return http.request.render("mymodule.index",
            {'fruits': fruits.search([])})

```

我们知道 `demo.xml` 里面定义的数据都放入全局环境里面去了。这里具体的细节还需要进一步讨论, 不过我们可以猜测通过 `http.request.env` 可以来引用这个全局环境的字典

值，来查找'mymodule.fruits' 的值，然后对这个值进行了 `search` 操作，这里具体的细节还需要进一步讨论，但我们可以猜得其最后返回的是一个列表值，然后里面存储的数据都是 `Fruit` 模型建立的对象。

而 `views/mymodule_templates.xml` 改成这样子了：

```
<openerp>
  <data>
    <template id="index">
      <title>MyModule</title>
      <t t-foreach="fruits" t-as="fruit">
        <p><t t-esc="fruit.id"/><t t-esc="fruit.name"></t></p>
      </t>
    </template>
  </data>
</openerp>
```

这里引用了 `fruit` 的 `id` 属性和 `name` 属性。我们可以看到这样的输出结果：



7apple

8banana

9pear

刚开始 `id` 是从 1,2,3 开始的，因为我运行过几次，在这里成 7,8,9 了。

哦，最后我们还需要修改一个东西：

### 5.3.4 security

`security` 文件夹里面已经有一个 `ir.model.access.csv` 文件了，其对模型的访问权限进行管理，大致修改内容如下：

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
access_mymodule_fruits,mymodule.fruits,model_mymodule_fruits,,1,0,0,0
```

目前还不太懂，看到 `perm_read` 那一列，要设置为 `1`，这样所有用户可读。然后一些名字是根据你定义的模型的名字变化的。

### 5.3.5 美化网页

Odoo 框架里面已经内置了一个帮助你设计网页的 `website builder` 模块，然后这里我们可以通过调用这个模块的结构来简单看一下。

首先是 `__openerp__.py` 的“depends”属性修改一下：

```
'depends': ['website'],
```

然后是 `controllers` 的 `main.py` 那里加上 `website=True` 这个设置。

```
@http.route('/mymodule/mymodule/', auth='public', website=True)
```

最后是模块文件那里修改为：

```
<openerp>
<data>
  <template id="index">
    <t t-call="website.layout">
      <t t-set="title">MyModule</t>
      <div class="oe_structure">
        <div class="container">
          <t t-foreach="fruits" t-as="fruit">
            <p><t t-esc="fruit.id"/><t t-esc="fruit.name"></t></p>
          </t>
        </div>
      </div>
    </t>
  </template>
</data>
```

```
</openerp>
```

请读者自己试着运行一下，看看效果，然后里面还有很多其他的设置功能等等。这里的细节先略过了。

## 5.4 加分项：通过 **pgadmin3** 来查看数据库

**pgadmin3** 是针对 PostgreSQL 数据库很有名的一个管理工具，里面的经很多，这里只是简单谈论一下。

### 5.4.1 安装

在 **ubuntu** 下可以用 **apt-get** 简单安装之。

```
sudo apt-get install pgadmin3
```

### 5.4.2 连接服务器

刚进入软件需要连接服务器，如下图所示：

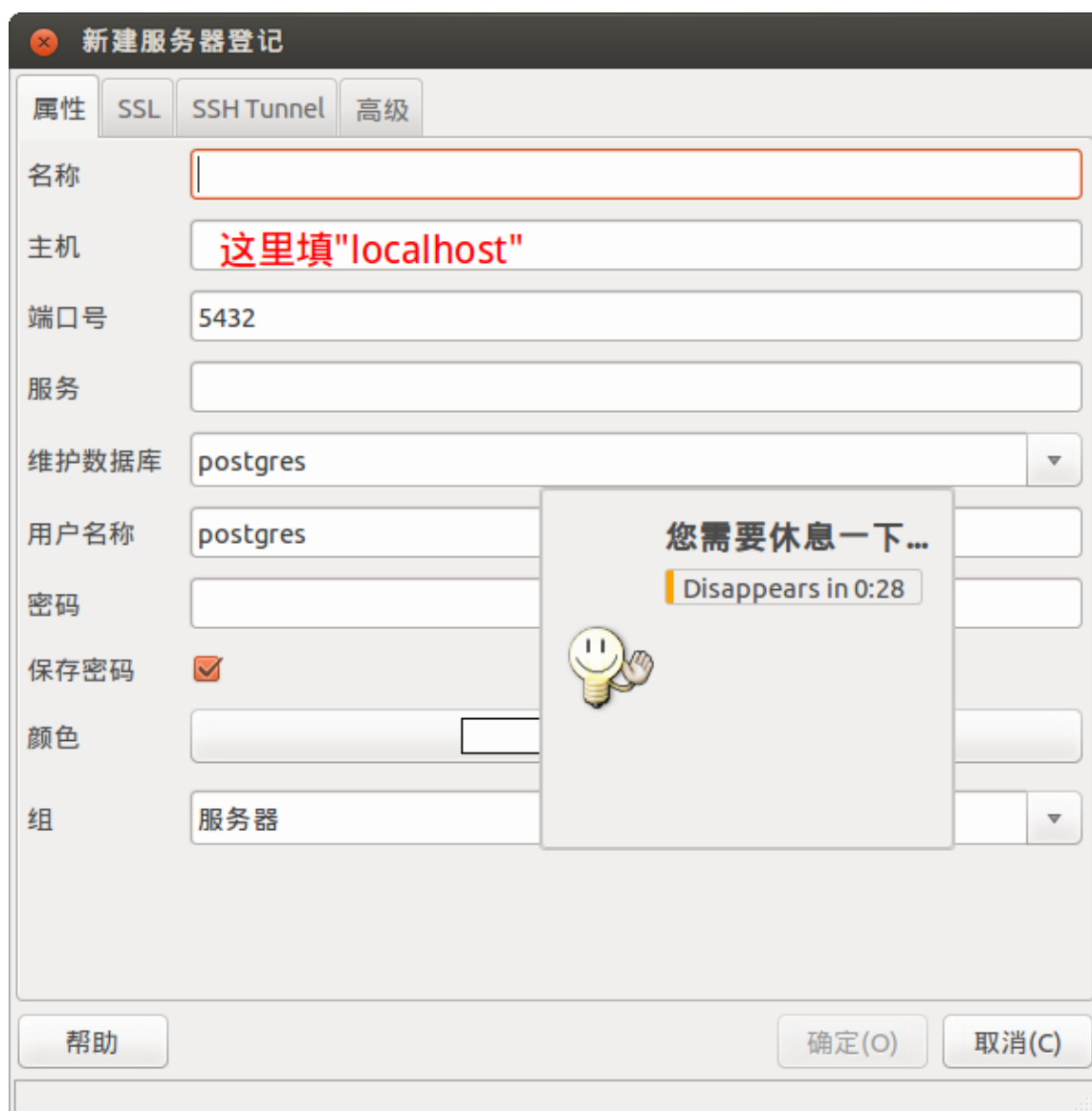


图 5.2: pgadmin3 连接服务器

目前我已经确认的就是名称随意填，然后主机不能填 127.0.0.1，而只能填“localhost”这个字符串。然后后面的应该不用更改什么了，如果你给你的 postgres 数据库设置密码了，那么这个密码也需要填上。

连接好服务器了，我们就可以双击或点击查看你的 PostgreSQL 数据库的信息了。

### 5.4.3 图形化查询

在工具那里有很多有用的工具，比如查看服务器状态工具等。然后我们点击查询工具，会弹出一个窗口，看到图形化查询那个子选单，如下图所示：



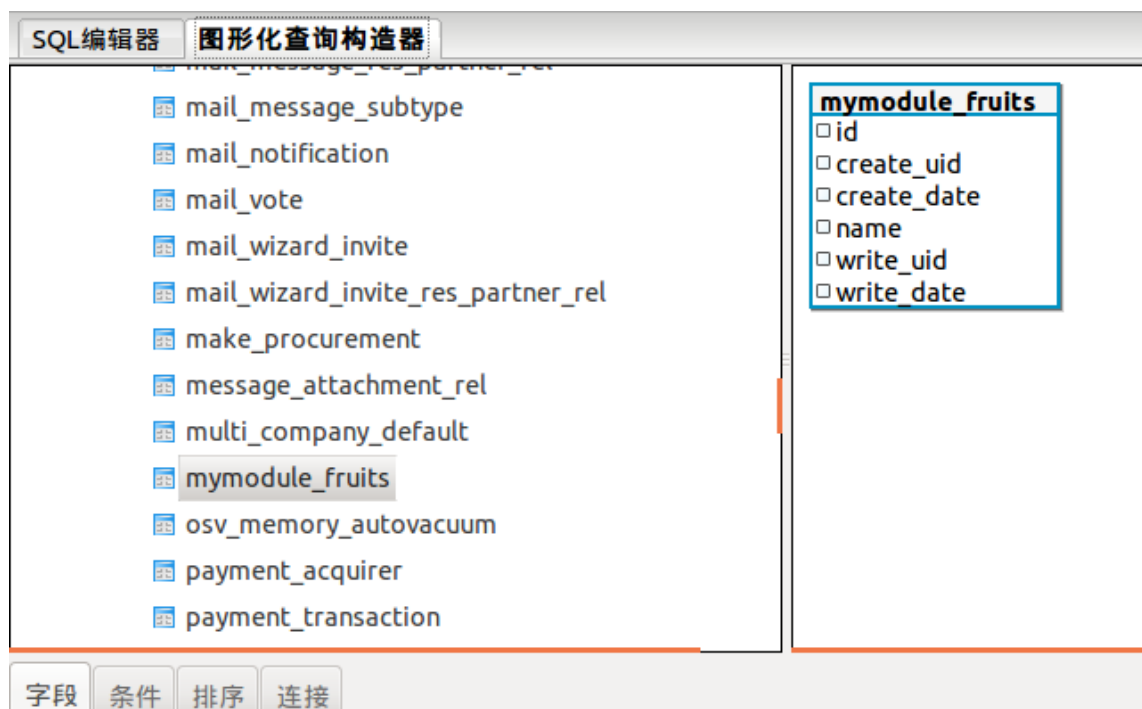


图 5.3: pgadmin3 图形化查询

这里我找到了前面我们新建的那个 `fruits` 模型，可以看到所有的 `fruits` 模型是放在一个名字叫做 `mymodule_fruits` 的 SQL 表格里的。然后它的表头有: `id`, `create_uid`, `create_date`, `name` 等。

然后我们切换到 SQL 编辑器子选单。这个工具会根据你的图形化查询选择结果自动生成对应的 SQL 查询语句：

```
SELECT
*
FROM
public.mymodule_fruits;
```

所以在我们这个 `odoo` 数据库里面，我们新建的模型 `fruits` 的各个对象，具体存入的 `table` 名是 `public.mymodule_fruits`。

然后我们点击查询 → 执行，来具体执行这个 SQL 语句，输出结果如下所示：

输出窗口						
数据输出 解释 消息 历史						
	id integer	create_uid integer	create_date timestamp without time zone	name character varying	write_uid integer	write_date timestamp without time zone
1	7	1	2015-05-08 03:36:14.734038	apple	1	2015-05-08 06:03:05.324781
2	8	1	2015-05-08 03:36:14.734038	banana	1	2015-05-08 06:03:05.324781
3	9	1	2015-05-08 03:36:14.734038	pear	1	2015-05-08 06:03:05.324781



## ODOO 开发基础：请假模块第一谈

### 6.1 纯理论讨论

在实际编写前先谈谈理论，这部分理论讨论非常有用，对于具体编写模块的时候你清楚自己在感谢什么很有帮助。感谢老肖的《OpenERP 应用和开发基础》一书，该书第六章对我帮助很大。

首先我们需要一个菜单，那么这个菜单在 Odoo 框架中是如何生成的呢？前面谈到 Odoo 的模型具体的对象实际上就是 SQL 表格的一条记录，而 Odoo 框架具体显示的菜单也是一个 Odoo 中的一个对象，其对应的表格是 `ir_ui_menu`，其在 xml 中的声明是通过 `menuitem` 标签来完成的，具体细节等下再讲。然后菜单需要连接一个动作，这样用户点击这个菜单的时候，这个动作将会触发。

这些动作对象（和窗口操作相关的）是存放在 `ir_act_window` 表格中的。动作触发之后接下来是要处理视图问题，首先根据 `ir_act_window_view` 表格来找到具体关联的某个视图对象，具体某个视图对象是存放在 `ir_ui_view` 表格中的。然后根据具体关联的某个模型的某个对象的具体的值来构建出显示画面。

具体研究对象的模型，视图，菜单，动作等，这些实际上都是 Odoo 里面的模型，也就是具体对象的值是存放在某个具体的 SQL 表格里的，然后程序完成一系列的索引，取值等操作，并最终生成显示结果，这大概就是 Odoo 框架里面发生的故事概貌了。

按照上面的讨论，等下我们的工作有：

1. 具体研究对象的模型，这里是请假单模型，然后请假单模型里面应该有的 field 有：申请人，请假天数，开始休假日期，请假事由。

2. 构建菜单对象。
3. 构建动作对象，并与具体的某个菜单关联起来。
4. 构建视图对象。

`__init__.py` 文件内容如下:

```
# -*- coding: utf-8 -*-  
  
from .models import main_model
```

`__openerp__.py` 文件内容如下:

```
# -*- coding: utf-8 -*-  
{  
    'name': "qingjia",  
  
    'summary': """  
        请假模块，提供请假功能  
        """,  
  
    'description': """  
        请假模块，提供请假功能。  
        """,  
  
    'author': "wanze",  
    'website': "http://www.yourcompany.com",  
  
    # Categories can be used to filter modules in modules listing  
    # Check https://github.com/odoo/odoo/blob/master/openerp/addons/base/module/module_data.xml  
    # for the full list  
    'category': 'Test',  
    'version': '0.1',  
  
    # any module necessary for this one to work correctly  
    'depends': ['base'],
```

```

# always loaded
'data': [
    'security/ir.model.access.csv',
    'views/views.xml',
],
# only loaded in demonstration mode
'demo': [
    'demo.xml',
],
'application' : True,
}

```

首先我们来看下 `main_model.py` 文件里面定义的模型是怎样的:

## 6.2 定义模型

```

from openerp import models, fields, api

class Qingjd(models.Model):
    _name = 'qingjia.qingjd'

    name = fields.Many2one('res.users', string=" 申请人", required=True)
    days = fields.Float(string=" 天数", required=True)
    startdate = fields.Date(string=" 开始日期", required=True)
    reason = fields.Text(string=" 请假事由")

    def send_qingjd(self):
        self.sended = True
        return self.sended

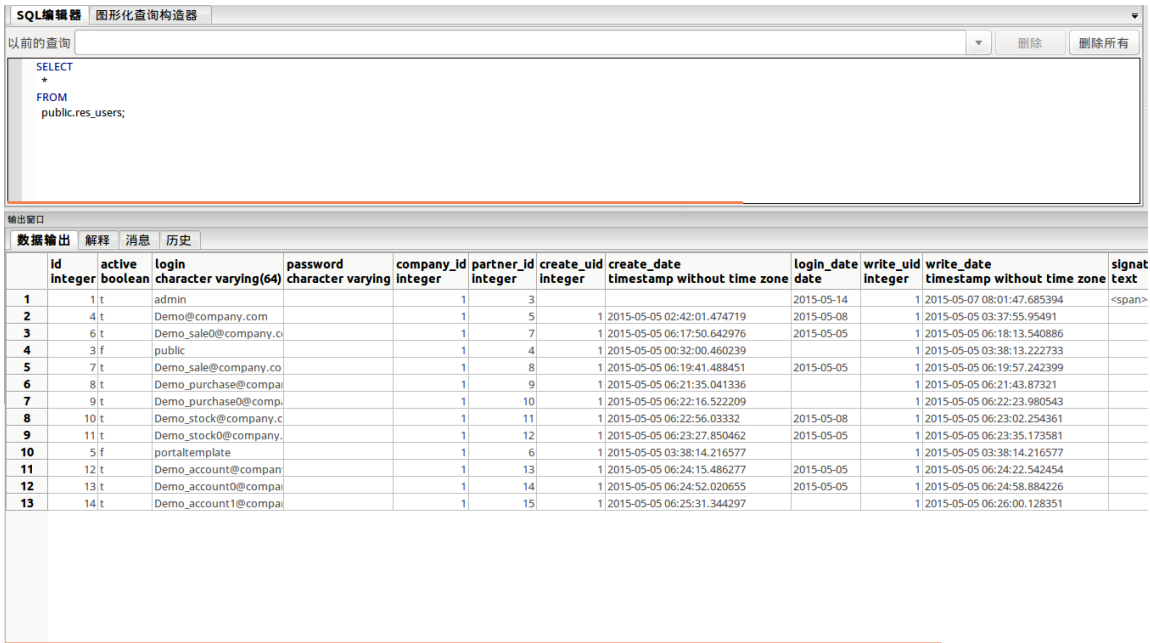
    def confirm_qingjd(self):
        self.state = 'confirmed'
        return self.state

```

这种模型定义语法结构我们大体是熟悉的了, 下面定义的两个方法等下会用到的, 等下再

谈。然后 Float 是浮点类型, Date 是日期类型这个是一目了然的, 然后 Text 是大段文本对象, string 是这个 field 的用户界面显示的文字, required 设置为 True 则该值为必填项。然后这个 Many2one 还有 res.users 是什么?

首先让我们看看 public.res.users 这个表格的值:



The screenshot shows the Odoo SQL Editor interface. At the top, there's a tab for 'SQL编辑器' and '图形化查询构造器'. Below it, a query is entered: `SELECT * FROM public.res_users;`. The bottom section, titled '输出窗口', displays the results of the query in a table format. The table has 13 columns: id, active, login, password, company\_id, partner\_id, create\_uid, create\_date, login\_date, write\_uid, write\_date, and signal. The data is listed in 13 rows, each representing a user record.

	id	active	login	password	company_id	partner_id	create_uid	create_date	login_date	write_uid	write_date	signal
	integer	boolean	character varying(64)	character varying	integer	integer	integer	timestamp without time zone	date	integer	timestamp without time zone	text
1	1	t	admin		1	3			2015-05-14	1	2015-05-07 08:01:47.685394	<span>
2	4	t	Demo@company.com		1	5	1	2015-05-05 02:42:01.474719	2015-05-08	1	2015-05-05 03:37:55.95491	
3	6	t	Demo_sale0@company.c		1	7	1	2015-05-05 06:17:50.642976	2015-05-05	1	2015-05-05 06:18:13.540886	
4	3	f	public		1	4	1	2015-05-05 00:32:00.460239		1	2015-05-05 03:38:13.222733	
5	7	t	Demo_sale@company.co		1	8	1	2015-05-05 06:19:41.488451	2015-05-05	1	2015-05-05 06:19:57.242399	
6	8	t	Demo_purchase@compa		1	9	1	2015-05-05 06:21:35.041336		1	2015-05-05 06:21:43.87321	
7	9	t	Demo_purchase@compa		1	10	1	2015-05-05 06:22:16.522209		1	2015-05-05 06:22:23.980543	
8	10	t	Demo_stock@company.c		1	11	1	2015-05-05 06:22:56.03332	2015-05-08	1	2015-05-05 06:23:02.254361	
9	11	t	Demo_account@compa		1	12	1	2015-05-05 06:23:27.850462	2015-05-05	1	2015-05-05 06:23:35.173581	
10	5	f	portaltemplate		1	6	1	2015-05-05 03:38:14.216577		1	2015-05-05 03:38:14.216577	
11	12	t	Demo_account@compa		1	13	1	2015-05-05 06:24:15.486277	2015-05-05	1	2015-05-05 06:24:22.542454	
12	13	t	Demo_account0@compa		1	14	1	2015-05-05 06:24:52.020655	2015-05-05	1	2015-05-05 06:24:58.884226	
13	14	t	Demo_account1@compa		1	15	1	2015-05-05 06:25:31.344297		1	2015-05-05 06:26:00.128351	

图 6.1: res.users 表格

这里 Many2one 的意思像是我从很多 (Many) 相同模型的对象中取一个 (one) 的意思。等下我们会看到一个下拉选单。更多细节需要深入学习 Odoo ORM 的 API 细节。

### 6.3 加入菜单

接下来的工作就是在 views/views.xml 文件里面定义具体的菜单对象。

代码第一版如下所示:

```
<?xml version="1.0"?>
<openerp>
<data>
    <record id="action_qingjia_qingjd" model="ir.actions.act_window">
        <field name="name"> 请假单</field>
        <field name="res_model">qingjia.qingjd</field>
        <field name="view_mode">tree,form</field>
    </record>
```

```

<menuitem id="menu_qingjia" name=" 请假" sequence="0"></menuitem>
<menuitem id="menu_qingjia_qingjiadan" name=" 请假单" parent="menu_qingjia"></menuitem>
<menuitem id="menu_qingjia_qingjiadan_qingjiadan" parent="menu_qingjia_qingjiadan" action="a

</data>
</openerp>

```

这种 record 语法我们已经有所熟悉了，然后 model 是 `ir.actions.act_window`，我们可以在源码 `openerp→addons→base→ir` 中找到 `ir_actions.py` 文件，然后有下面的代码：

```

class ir_actions_act_window(osv.osv):
    _name = 'ir.actions.act_window'
    _table = 'ir_act_window'

```

我们可以看到其对应的正是表格 `ir_act_window`。

`field` 是用来填充具体表格的某个列值的，我们还可以使用如下简便的语法：

```

<act_window id="action_qingjia_qingjd"
    name=" 请假单"
    res_model="qingjia.qingjd"
    view_mode="tree,form" />

```

这里的标签 `act_widow` 还不清楚是在那里规定的，然后下面具体的菜单对象也简便使用了 `menuitem` 标签。

我们可以在 `openerp→addons→base→ir` 中找到 `ir_ui_menu.py` 文件，有如下代码：

```

class ir_ui_menu(osv.osv):
    _name = 'ir.ui.menu'

```

可以看到菜单对象对应的是 `ir_ui_menu` 表格——和数据模型一样的映射法则，`_name` 的点号变成下划线。至于菜单对象为何对应 `menuitem` 标签还不清楚那里固定的。

这两个对象具体的一些属性说明一下：

### 6.3.1 act\_window 的属性

**name** 具体 `act_window` 动作在 UI 中显示的名字（类似于 QT 中动作作为菜单中的项目的情况）。

**res\_model** `act_window` 动作对应的某个数据模型（这里动作和数据模型关联在一起了）

**view\_mode** `act_window` 动作打开后支持的视图模式。

### 6.3.2 menuitem 的属性

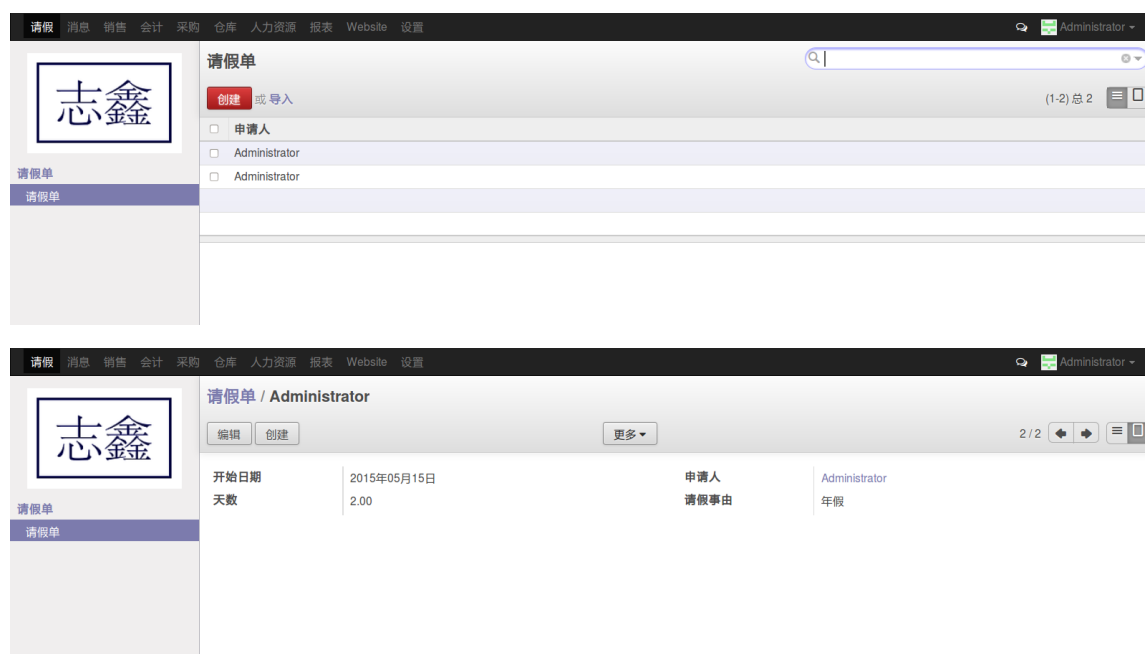
**name** 具体这个菜单在视图中显示的名字。

**sequence** 是显示排序（还不太懂）。

**parent** 是本菜单的父菜单。如果是子菜单则需要指定，只有顶级菜单不需要指定。

**action** 指定本菜单连接的动作。如果连接动作了那么 **name** 属性可以不用指定了，系统会直接引用动作的 **name** 属性的。这里菜单和某个动作关联起来了。和前面联系起来，那么就是具体某个子菜单和某个数据模型关联起来了。

这样现在你应该看到类似如下的视图了：



## 6.4 视图优化

最主要的三个视图是 **list** 或者 **tree** 视图；表单 **form** 视图和 **search** 搜索视图。视图都属于 `ir.ui.view` 模型



### 6.4.1 修改 tree 视图

下面定义了一个自己的 tree 视图:

```
<record id="qingjia_qingjd_tree" model="ir.ui.view">
<field name="name">qing jia dan tree</field>
<field name="model">qingjia.qingjd</field>
<field name="arch" type="xml">
    <tree>
        <field name="name"/>
        <field name="startdate"/>
        <field name="days"/>
    </tree>
</field>
</record>
```

这里的 **name** 属性是本视图的名字，似乎没什么意义。然后 **model** 属性很重要，前面子菜单关联到了某个动作，然后某个动作关联到了某个数据模型了，这里就是将这个视图和某个模型关联起来了。

下面的 **arch** 这个格式还不清楚有什么用，但必须这么写:

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

然后接下来定义 **tree** 视图，又叫列表视图。其中的 **field** 就是对应的具体那个数据模型的 **field**，加入谁就要显示谁。

### 6.4.2 修改 form 视图

下面定义一个自己的 form 视图:

```
<record id="qingjia_qingjd_form" model="ir.ui.view">
<field name="name">qing jia dan form</field>
<field name="model">qingjia.qingjd</field>
<field name="arch" type="xml">
    <form>
```

```
<sheet>
<label for="name"/> <field name="name"/>
<label for="days"/><field name="days"/>
<label for="startdate"/><field name="startdate"/>
<label for="reason"/><field name="reason"/>
</sheet>
</form>
</field>
</record>
```

其他都类似上面的，不赘述了。

重点在 **form** 标签里面。这里引入了 **sheet** 布局，然后 **label** 加入标签 **field** 引入具体属性。

然后我们还可以使用 **header** 标签引入表单视图的头部分。头部分里面一般放着一些按钮动作。比如：

```
<header>
  <button name="send_qingjd" type="object"
    string=" 发送" class="oe_highlight" />
  <button name="confirm_qingjd" type="object"
    string=" 确认" />
</header>
```

这里 **string** 是这个按钮具体显示的字符，然后 **name** 是这个按钮具体应该执行的动作（对应本模型的该名字的方法），**class="oe\_highlight"** 让按钮变为红色突出显示。

## 使用 **group** 布局

此外还可以使用 **group** 布局，**group** 布局还不太懂，其中可以引入 **string** 属性，等下可以作为 **group** 的标题显示出来。

这是我的这个简单的请假单布局

```
<sheet>
  <group name="group_top" string=" 请假单">
    <field name="name"/>
    <field name="days"/>
```

```

        <field name="startdate"/>
        <field name="reason"/>
    </group>
</sheet>

```

## 6.5 完整的 views.xml

至此完整的 views.xml 文件如下所示:

```

<?xml version="1.0"?>
<openerp>
<data>

<!--
    打开请假单动作
-->
    <act_window id="action_qingjia_qingjd"
        name=" 请假单"
        res_model="qingjia.qingjd"
        view_mode="tree,form" />

<!--
    表单视图
-->
    <record id="qingjia_qingjd_form" model="ir.ui.view">
        <field name="name">qing jia dan form</field>
        <field name="model">qingjia.qingjd</field>
        <field name="arch" type="xml">
            <form>
                <header>
                    <button name="send_qingjd" type="object"
                        string=" 发送" class="oe_highlight" />
                    <button name="confirm_qingjd" type="object"
                        string=" 确认" />
                </header>

```

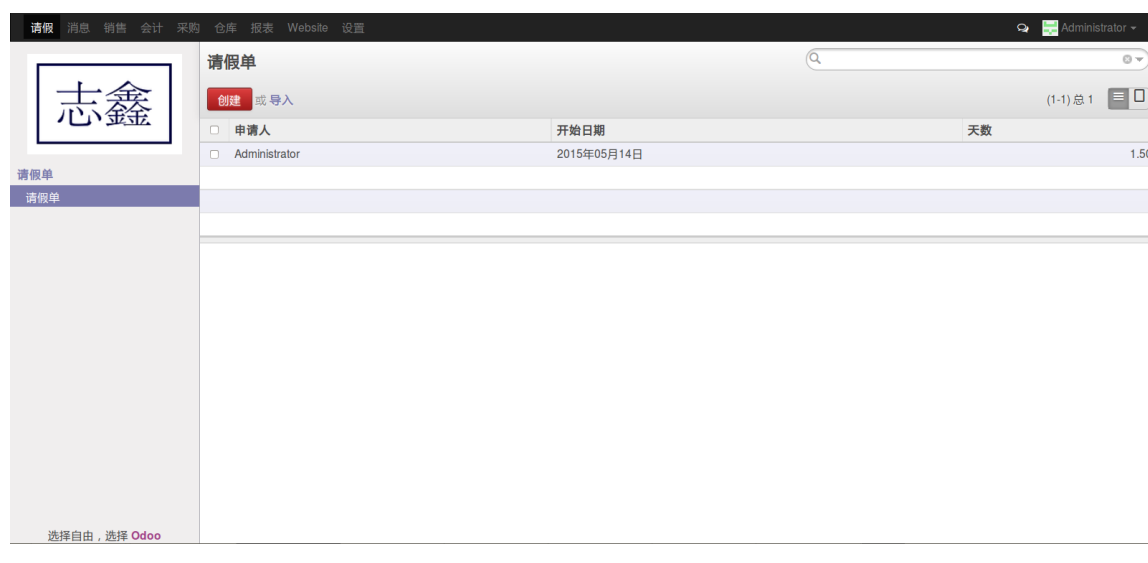
```

        <sheet>
            <group name="group_top" string=" 请假单">
                <field name="name"/>
                <field name="days"/>
                <field name="startdate"/>
                <field name="reason"/>
            </group>
        </sheet>

    </form>
</field>
</record>
<!--
tree 视图
-->
    <record id="qingjia_qingjd_tree" model="ir.ui.view">
        <field name="name">qing jia dan tree</field>
        <field name="model">qingjia.qingjd</field>
        <field name="arch" type="xml">
            <tree>
                <field name="name"/>
                <field name="startdate"/>
                <field name="days"/>
            </tree>
        </field>
    </record>
<!--
加入菜单
-->
    <menuitem id="menu_qingjia" name=" 请假" sequence="0"></menuitem>
    <menuitem id="menu_qingjia_qingjiadan" name=" 请假单" parent="menu_qingjia"></menuitem>
    <menuitem id="menu_qingjia_qingjiadan_qingjiadan" parent="menu_qingjia_qingjiadan" action="a
</data>
</openerp>

```

至此显示效果如下所示:



### 请假单

申请人

天数

开始日期

请假事由

Administrator

1.50

2015年05月

comodel\_name 第一个参数 res.users  
设置目标模型的名字。

## 6.6 给模块加个图标

模块 `static→description` 的 `icon.png` 文件就对应模块的图标。把它设置好你的模块就有一个图标了。



## ODOO 开发基础: 工作计划模块第一谈

这个例子来自 Daniel Reis 的《Odoo Development Essentials》一书，之所以也加进来是因为我觉得学习框架是例子越多越好，然后这本书我是在 [这个网站](#) 限时观看的，也是为了保留有价值的信息吧。

我们这次要创建的模块的功能是进行工作计划管理，也就是常说的“to do task”，

首先是 `__init__.py` 文件：

```
# -*- coding: utf-8 -*-  
  
from .models import main_model
```

然后是 `__openerp__.py` 文件：

```
# -*- coding: utf-8 -*-  
{  
    'name': "todo task app",  
  
    'summary': """  
        工作计划管理系统  
        """,  
  
    'description': """  
        工作计划管理系统：安排你的工作计划。  
        """
```

```

"""
    'author': "wanze",
    'website': "http://www.yourcompany.com",

    # Categories can be used to filter modules in modules listing
    # Check https://github.com/odoo/odoo/blob/master/openerp/addons/base/module/module_data.xml
    # for the full list
    'category': 'Test',
    'version': '0.1',

    # any module necessary for this one to work correctly
    'depends': ['mail'],

    # always loaded
    'data': [
        'security/ir.model.access.csv',
        'views/views.xml',
    ],
    # only loaded in demonstration mode
    'demo': [
        # 'demo.xml',
    ],
    'application' : True,
}

```

注意到 **depends** 设置为了 **mail** 模块，因为等下 **to do task** 子菜单要在消息子菜单中显示的，其是由 **mail** 模块设置的。然后同样加载了 **views.xml** 文件，还有设置访问权限的 **ir.model.access.csv** 文件等下再说。

然后我们再来看 **main\_model.py** 文件

```

from openerp import models, fields, api

class TodoTask(models.Model):
    _name = 'todo.task'

```



```

name = fields.Char('Description', required=True)
is_done = fields.Boolean('Done?')
active = fields.Boolean('Active?', default=True)

@api.one
def do_toggle_done(self):
    self.is_done = not self.is_done
    return True

@api.multi
def do_clear_done(self):
    done_recs = self.search([('is_done', '=', True)])
    done_recs.write({'active': False})
    return True

```

读者现在应该很清楚这些代码在做什么了，其中 `fields.Boolean` 定义了一个布尔值字段，然后我们看到第一个可选参数默认就是 `string`，就可以直接写上。然后 `default` 是来设置该字段的默认值的。

下面定义的两个方法和等下 `form` 视图下的两个按钮相关，这里用到了 Odoo 新的 ORM API，使用了 `@api.one` `@api.multi` 这样的装饰器。值得一提的是，可被用户调用的方法，比如这里的按钮是被用户点击才调用的，都需要一个返回值，否则 XMLRPC 协议无法正常工作，实在没啥好返回的，就 `return True`。

`@api.multi` 默认的装饰器是这个，没有自动迭代 `recordset`，因为它默认接受的 `self` 就是 `recordset` 对象（所有 `recordset` 就是指相同模型下的所有对象，或者说同一 SQL 表格下的所有记录。）。

`@api.one` `@api.one` 装饰器将会自动产生一个迭代动作，具体是指迭代某一 `recordset`，然后其内的 `self` 就是一个 `record` 也就是该模型下 SQL 表格的一条记录。然后 `@api.one` 返回的是一个列表值，某些网络客户端可能并不支持这点。所以还是尽量少用 `@api.one`。

具体 `views.xml` 文件的内容如下所示：

```

<?xml version="1.0"?>
<openerp>
<data>

```

```
<!-- Action to open To-do Task list -->
<act_window id="action_todo_task"
name="To-do Task"
res_model="todo.task"
view_mode="tree,form" />

<record id="view_form_todo_task" model="ir.ui.view">
<field name="name">To-do Task Form</field>
<field name="model">todo.task</field>
<field name="arch" type="xml">

<form>
<header>
    <button name="do_toggle_done" type="object"
        string="Toggle Done" class="oe_highlight" />
    <button name="do_clear_done" type="object"
        string="Clear All Done" />
</header>
<sheet>
    <group name="group_top">
        <group name="group_left">
            <field name="name"/>
        </group>
        <group name="group_right">
            <field name="is_done"/>
            <field name="active" readonly="1" />
        </group>
    </group>
</sheet>
</form>

</field>
</record>
```

```

<record id="view_tree_todo_task" model="ir.ui.view">
  <field name="name">To-do Task Tree</field>
  <field name="model">todo.task</field>
  <field name="arch" type="xml">
    <tree colors="gray:is_done==True">
      <field name="name"/>
      <field name="is_done"/>
    </tree>
  </field>
</record>

<record id="view_filter_todo_task" model="ir.ui.view">
  <field name="name">To-do Task Filter</field>
  <field name="model">todo.task</field>
  <field name="arch" type="xml">
    <search>
      <field name="name"/>
      <filter string="Not Done"
        domain="[('is_done','=',False)]"/>
      <filter string="Done"
        domain="[('is_done','!=',False)]"/>
    </search>
  </field>
</record>

<!-- Menu item to open To-do Task list -->
<menuitem id="menu_todo_task"
  name="To-Do Tasks"
  parent="mail.mail_feeds"

  sequence="20"
  action="action_todo_task" />

```

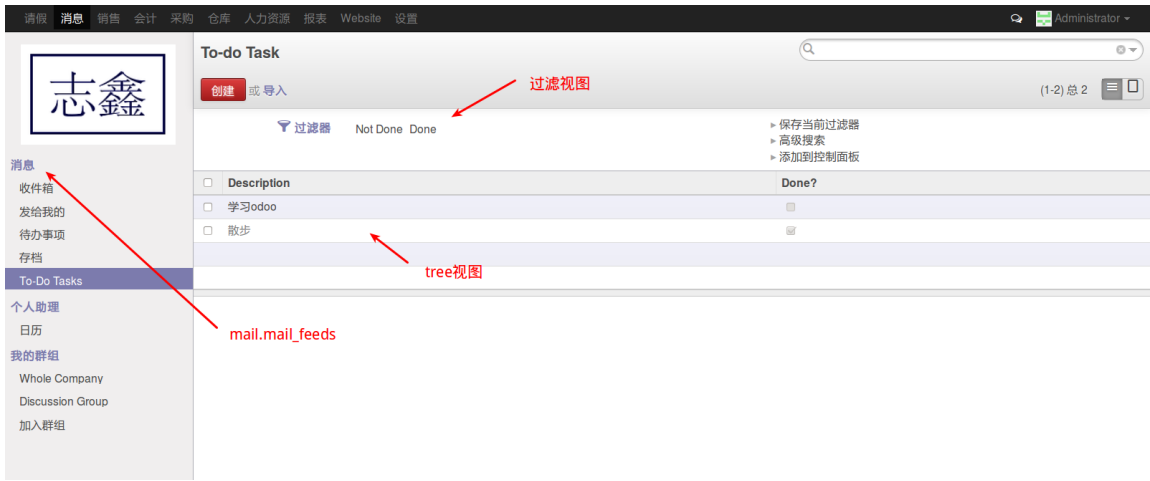
```
</data>

</openerp>
```

有了前面的基础，这个文件虽然看上去有点复杂，但我们应该是已经能够看出端倪出来了。首先我们不管视图那块，看到 `act_window` 和 `menuitem`，我们看到这个菜单项设置的父菜单是 `mail.mail_feeds`，

```
parent="mail.mail_feeds"
```

因为对 Odoo 官方模块还不是很熟悉，但我们根据后面的显示效果，如下图所示：



我们可以推断这个父菜单 `mail.mail_feeds` 就是上图箭头所指的那个菜单。

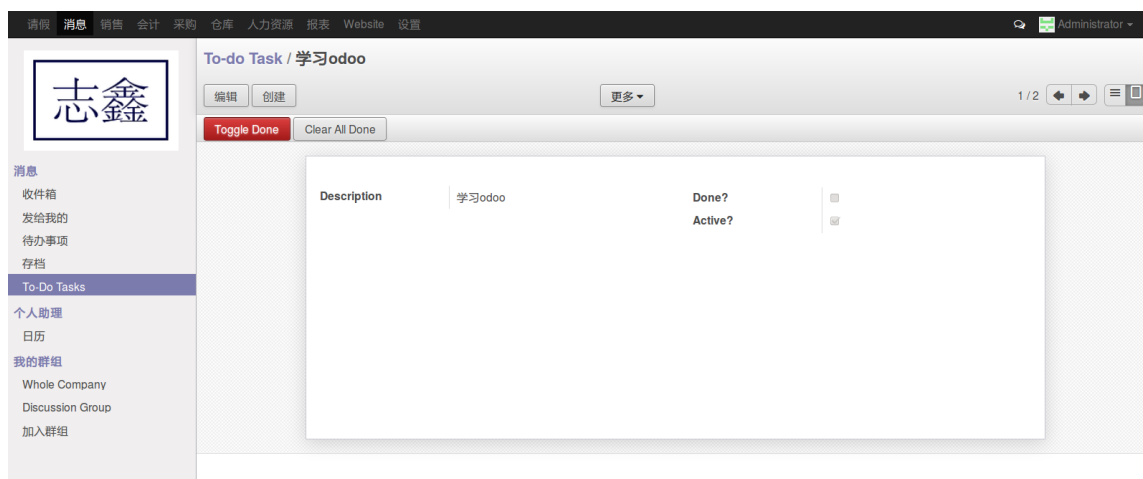
然后我们看到 `tree` 视图，其中有这么一行：

```
<tree colors="gray:is_done==True">
```

这种语法还不太熟悉，但大体意思就是如果该模型下的某个对象 `is_done` 的值为 `True`，则颜色设置为灰色。

然后值得一提的是，这里的 `search` 视图，并不需要在前面的动作对象的 `view_mode` 属性上加上，其更像是 `tree` 视图的增强功能，在搜索框那里，你点击才能看到。代码中唯一的难点就是 `domain` 过滤器的语法，这里先略过，后面再讨论吧。

然后我们再看到 `form` 视图，具体显示效果如下所示：



```
<sheet>
  <group name="group_top">
    <group name="group_left">
      <field name="name"/>
    </group>
    <group name="group_right">
      <field name="is_done"/>
      <field name="active" readonly="1" />
    </group>
  </group>
</sheet>
```

这里代码如何利用 **group** 来布局要好好体会一下，然后 **field** 标签这里又出现了一个新的属性 **readonly**，这样这个字段的值就不能更改了。

## 7.1 数据访问权限管理

Odoo 对于其内的数据访问权限管理有两种机制：一种是模型访问权限管理 (**access rule**)；第二种是记录规则管理 (**record rule**)。其中 **record rule** 可以看作在 **access rule** 之上的更进一步细化。如果什么访问规则都不设定的话，那么默认只有 **Administrator** 才能访问这个模型的数据，其他的用户都不能访问。Odoo 的安全机制是通过群组 (**group**) 来管理的，然后 **record rule** 对 **Administrator** 用户是无效的，而 **access rule** 还有效。

<https://www.odoo.com/forum/help-1/question/is-it-possible-to-manage-record-b>

### 7.1.1 access rule

访问权限控制一般是用 `security` 文件夹下的 `ir.model.access.csv` 文件来管理的。这个 `csv` 文件的表头是：

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
```

我们这里定义的模型名字是 `todo.task`，其对应的 `model_id` 就是 `model_todo_task`。这个是规定死了的，然后其他的 `id`, `name` 名字可随意。一般 `id` 就是在模型名字前加个 `access` 即 `access_todo_task`，然后 `name` 更随意了，似乎我看到空格都可以的，但一般就简单把对应的模型的名字放进去即可，即 `todo.task`。然后第四个是群组 `id`，接下来就是读权限，写权限，创建权限和删除权限，用 `0` 表示无权限，`1` 表示有权限，具体要根据需要来设置权限。

下面是 `todo_app` 模块的 `ir.model.access.csv` 详情：

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
access_todo_task,todo.task,model_todo_task,base.group_user,1,1,1,1
```

The default policy is DENY

这里的 `base.group_user` 群组参考资料 4 说雇员群组属于 `base.group_user` 群组，这里不太清楚，如果实在不确定这个值省略不填也是可以的。

### 7.1.2 record rule

这样设置之后每一个登录 Odoo 用户写的 `todo task` 其他人都可以看到，这显然不太好。进一步我们可以用 `record rule` 来对具体的 `record` 记录进行筛选。

在 `security` 文件夹里新建一个 `todo_record_rules.xml` 文件，文件内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<openerp>
  <data noupdate="1">
    <record id="todo_task_user_rule" model="ir.rule">
      <field name="name">ToDo Tasks only for owner</field>
      <field name="model_id" ref="model_todo_task"/>
      <field name="domain_force">[('create_uid','=',user.id)]
    </field>
    <field name="groups" eval="[ (4,ref('base.group_user')) ]"/>
  </record>
</data>
</openerp>
```

```
</record>
</data>
</openerp>
```

这些 record rule 记录是 `ir.rule` 模型，存储在 `public.ir_rule` 表格里面的。`name` 属性就不多说了，应该很熟悉了。然后 `model_id` 属性对应的就是某个模型，而 `domain_force` 就是对该模型中所有记录进行某种过滤操作，

The default policy is ALLOW,so by default any operation will be refused if the user does not explicitly have the right to perform it via one of her groups' access rights.

The default policy is ALLOW, so if no rule exists for a given model, all documents of that model may be accessed by users who have the necessary access rights. 然后 `groups` 字段还看不太懂。我看了一下 `ir_rule` 表格，似乎没有 `groups` 这个属性，这里的细节先略过吧。

`data` 标签这里有个 `noupdate` 属性，如果设置为“1”，则意思是本模块升级不更新本数据，如果设置为“0”则更新本数据。

然后我还发现一个问题：那就是我中途试验过一个不加载这个 `xml` 文件的版本，数据库里面还是保存着这条记录，然后我手动删除 `SQL` 数据库的这条记录才行，后来又试验加载这个 `xml` 文件的版本，数据库这个权限记录加载不上去。然后我发现如果删除模块，则数据库里面的这条记录才会被删除了。然后再重新安装本模块，则这条权限记录才会被更新上去。也就是说目前的 Odoo 框架升级对于 `record rule` 支持并不是很好，在这方面要小心。

经过这样的配置之后，除了 `Administrator` 用户可以看到所有人的 `todo task` 之外，其他人都只能看到和编辑自己的 `todo task` 了。





## 扩展现有模块—继承机制

即使是对于现有的模块，推荐的做法也是通过新建一个模块来达到扩展和修改现有模块的目的。具体方法就是在 `python` 中的类里面使用 `_inherit` 属性。这标识了将要扩展的模块。新的模型继承了父模型的所有特性，我们只需要声明一些我们想要的修改就行了。通过这种继承机制的修改可从模型到视图到业务逻辑等对原模块进行全方位的修改。

实际上，Odoo 模型在我们定义的模型之外，它们都在注册中心注册了的，所谓全局环境的一部分，可以用 `self.env[model name]` 来引用之。比如要引用 `res.partner` 模型，我们就可以写作 `self.env['res.partner']`。

### 8.1 给模块增加 field

如下代码就是首先通过 `_inherit` 继承原模块，然后再增加一些 field:

```
from openerp import models, fields, api

class TodoTask(models.Model):

    _inherit = 'todo.task'

    user_id = fields.Many2one('res.users', string='Responsible')
    date_deadline = fields.Date('Deadline')
```

关于 `res.users` 和 `res.partner` 具体是雇员还是合作伙伴什么的，这个以后再摸清楚，这里先简单将其看作一个 SQL 表格，然后 `Many2one` 前面讲过了就是根据某个给定的 SQL 表格来生成一个下拉选单，具体是引用的该 SQL 表格的那个表头属性，这里应该还有一个细节讨论。

不管怎么说，现在我们通过新建一个模块 `todo_user`，如前面描述的将模块设置配置好之后，原模块 `todo_app` 的 `todo.task` 模型就增加了新的两个 `field` 了，也就是两个新的表头了。

## 8.2 修改已有的 field

按照上面的继承机制，我们可以如上类似处理，只修改你希望更改的某个 `field` 的某个属性即可。如下：

```
name = fields.Char(help="can I help you")
```

这样原模型的 `namefield` 额外增加了 `help` 帮助信息了。

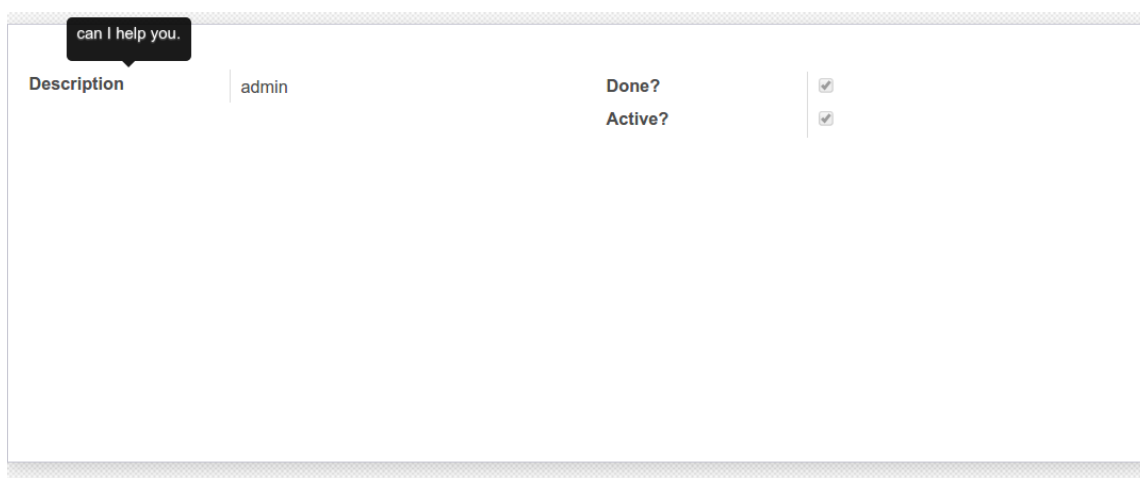


图 8.1: help 帮助信息

## 8.3 重载原模型的方法

读者一定已经想到了，类似的在这种继承机制下，可以通过重写原模型的方法来重载该方法。事实上确实可以这样做，而这里要讲的是还有一种更加优雅的继承原模型的方法，那就是通过 `super()` 来调用父类的方法<sup>1</sup>。

<sup>1</sup>在 `python3` 中就可以直接使用 `super().what` 来引用父类的 `what` 属性，在 `python2` 中需要加上 `super(TodoTask, self)` 这两个参数，也就是具体父类的类名和 `self`。

首先我们看到下面这个例子:

```
@api.multi
def do_clear_done(self):
    domain = [('is_done', '=', True),
              '|', ('user_id', '=', 'self.env.uid'),
              ('user_id', '=', False)]
    done_recs = self.search(domain)
    done_recs.write({'active': False})
    return True
```

这里涉及到 Odoo 新 API 的一些东西, 这里先浅尝辄止讲一下。

### 8.3.1 什么是 Recordset

Odoo8 引入了一种新的 ORM API, 老的 API 也兼容, 但推荐使用新的 API, 因为新的 API 更加简洁和方便。

首先是模型 (model), 其对应的就是 python 的类, 具体类的实例就是对应现实世界的某个对象。然后老式的简单 ORM 封装就是将这些类的具体某些数据对应到 SQL 的数据库的一条记录 (record) 中去。新的 API 引入一个核心的概念就是 Recordset, Recordset 是个什么东西呢? 就是前面讲的某一个模型 (类) 的所有对象 (具体的实例) 的集合就是一个 Recordset 对象。——这是 recordset 最大的情况, 一个重要的限定条件就是其内元素必定是相同模型的, 由这个最大的集合情况然后删除过滤掉一些元素 (记录) 之后仍然是 recordset 对象。

按照官方文档的描述是, 一个 Recordset 对象应该是已经排序了的同一模型的对象集合。他还指出虽然现在还可以存放重复的元素, 这个以后可能会变的。同时你从名字可能猜到这个 Recordset 对象应该支持集合的一些操作, 事实确实如此。

比如 Recordset 支持如下运算:

```
record in recset1      # include
record not in recset1  # not include
recset1 + recset2      # extend
recset1 | recset2      # union
recset1 & recset2      # intersect
recset1 - recset2      # difference
recset.copy()          # copy the recordset (not a deep copy)
```

上面的操作只有 + 还保留了次序，不过 `recordset` 是可以排序的，关于次序比如使用：

```
for record in recordset:
    print(record)
```

具体的次序是否像集合 `set` 一样是不一定的还是如何呢？这里需要进一步的讨论。

### 8.3.2 Odoo 里面的 domain 语法

本小节主要参考了 [这个网页](#)。

Odoo 里面的 `domain` 语法使用比较广泛，其就好像一个过滤器，应该对应的是 SQL 的 `SELECT` 语句。最基本的语句形式是 `[('field_name', 'operator', value)]`

`field_name` 必须是目标模型的有效 `field` 名字。

`operator` 比如是一个字符串，可用的值有：`=` `!=` `>` `>=` `<` `<=` `like` `ilike`，此外还有`"in"`，`"not in"`，`"parent_left"`，`"child_of"`，`"parent_right"`。这里的 `parent` 和 `child` 似乎是某种记录的关系，先暂时略过。其他的意义都是很明显的。

`value` 必须是和前面的 `field_name` 类型相同的某个值。

然后这些圆括号包围的基本语句可以用以下几个逻辑运算符连接：`&` `|` `!`，其中 `&` 是默认的逻辑运算连接符，也就是你看到两个圆括号表达式中间没有逻辑运算连接符，则要视作其间加入了 `&`。具体形式大概类似这样：

```
[('field_name1', 'operator', value), '!',
 ('field_name2', 'operator', value), '|',
 ('field_name3', 'operator', value), ('field_name4', 'operator', value)]
```

多个逻辑运算符的情况有点复杂，具体是！先解析，其只作用于后面的第一个元素；然后 `&` 和 `|` 作用于后面的两个元素。一个简单的解析步骤是先将 `!` 解析进去，比如是解析为不是等，然后再将 `|` 解析进去，相当于一个并联电路接进来，然后所有的过滤条件组成一个大的串联过滤线路。这样上面的表达式就解析为：

1 表达式 and 2 表达式否 and 3 表达式或 4 表达式

然后前面的那个 `domain`：

```
domain = [('is_done', '=', True),
 '|', ('user_id', '=', 'self.env.uid'),
```

```
('user_id', '=', False)]
```

应该解析为:

```
is_done 是 True and user_id 是 self.env.uid 或 user_id 是 False
```

### 8.3.3 recordset 的 search 方法

一个 recordset 对象调用其 search 方法还是返回一个 recordset 对象。

search 方法接受一个参数, 这个参数就是前面谈论的基于 Odoo domain 语法的过滤器表达式。

所以下面这个表达式:

```
self.env['res.users'].search([('login', '=', 'admin')])
```

的含义就是调用 res.users 这个表格或者说 recordset, 然后执行 search 方法, 具体选中的 record 是 login 这个字段等于 admin 的。

好了前面那个 do\_clear\_done 函数我们应该完全理解了, 首先 @api.multi 告诉我们这个函数里面的 self 是一个 recordset, 然后 domain 的语法是: is\_done 是 True 或说被勾选了, 然后要某该记录的 user\_id 等于当前用户的 id self.env.uid, 要某 user\_id 值为 False (不清楚什么情况)。

接下来执行 search 方法, 返回的 done\_recs 也是一个 recordset 对象, 对于这些 recordset 对象执行了 write 方法, 其接受一个字典值, 就是直接更改 SQL 表格里面的某个表头 (属性), 将其改为某个值。值得一提的是, recordset 调用 write 方法会将本 recordset 内所有的 record 都进行修改操作的。

前面讲到通过 super() 来继承修改原模型的某个方法, 请看下面的例子:

```
@api.one
def do_toggle_done(self):
    if self.user_id != self.env.user:
        raise Exception('Only the responsible can do this!')
    else:
        return super(TodoTask, self).do_toggle_done()
```

这里 @api.one 自动遍历目标 recordset, 然后方法里面的 self 就是一个 record。这

里程序的逻辑很简单，就是如果用户名不是当前登录用户（因为 **todo task** 管理只是自己管理自己的任务计划），那么将会报错。如果是那么就调用之前的方法。

## 8.4 视图 xml 文件的继承式修改

一个初步的继承式修改视图 xml 文件如下所示：

```
<?xml version="1.0"?>
<openerp>
<data>
  <record id="view_form_todo_task_inherited" model="ir.ui.view">
    <field name="name">Todo Task form – User extension</field>
    <field name="model">todo.task</field>
    <field name="inherit_id" ref="todo_app.view_form_todo_task"/>
    <field name="arch" type="xml">
      <field name="name" position="after">
        <field name="user_id" />
      </field>
      <field name="is_done" position="before">
        <field name="date_deadline" />
      </field>
      <field name="name" position="attributes">
        <attribute name="string">I have to...</attribute>
      </field>
    </field>
  </record>
</data>
</openerp>
```

我们可以看到其通过这样的语句：

```
<field name="inherit_id" ref="todo_app.view_form_todo_task"/>
```

对 xml 视图进行了继承。这里是要对 **from** 视图进行修改，就继承的原 **form** 视图的 **id**。

### 8.4.1 视图元素添加

首先我们来看视图元素的添加问题。Odoo 提供了这样的定位语法：

```
<field name="name" position="after">
    <field name="user_id" />
</field>
```

其具体对应的是所谓的 XPath 语法，比如 `<field name="is_done">` 对应的是：

```
//field[@name]='is_done'
```

除了 `field`，其他的 `tag` 如 `sheet`、`group` 等等都是可以用的，属性 `name` 最常使用，其他的属性也是可以用的。定位到具体的标签之后，需要使用 `position` 来指明插入点。

`inside` 默认的就是 `inside`，也就是插入定位标签之内。

`before` 插入定位标签之前。

`after` 插入定位标签之后。

`replace` 替换掉定位标签的元素，如果使用空内容，则就是删除原标签元素。

比如这个例子

```
<field name="name" position="after">
    <field name="user_id" />
</field>
<field name="is_done" position="before">
    <field name="date_deadline" />
</field>
```

的意思就是找到 `field name="name"` 的那个标签，然后在它的后面插入 `<field name="user_id" />`。

然后找到 `field name="is_done"` 的那个标签，在它的前面插入 `<field name="date_deadline" />`。

### 8.4.2 原视图元素属性修改

`position` 如果设置为 `attributes`，则可以具体对原标签元素的某个属性进行修改。

**attributes** 修改定位标签元素的某个属性。

比如这样:

```
<field name="name" position="attributes">
    <attribute name="string">I am going to</attribute>
</field>
```

再如:

```
<field name="active" position="attributes">
    <attribute name="invisible">1</attribute>
</field>
```

之前的 **active field** 没必要显示出来了, 可以将这个字段的 **invisible** 属性设置为 **1**, 让这个字段在视图上不显示即可。前面讲到 **replace** 说到可以删除某个标签元素, 但一般不建议这样做, 因为可能其他扩展模块又依赖这个标签元素。最好就是将它的 **invisible** 属性修改一下即可。

读者可以看到 之前那个 **form** 视图。

经过如上的修改, 现在成了这个样子了:

The screenshot shows a web-based form interface. At the top, there are two buttons: 'Toggle Done' (in red) and 'Clear All Done'. Below them is a table with four columns. The first column is labeled 'I am going to Responsible'. The second column contains the text '跑步' and 'Administrator' below it. The third column is labeled 'Deadline Done?'. The fourth column contains the date '2015年05月19日' and a small square checkbox below it.

## 8.5 多态继承

**\_inherit** 继承也可以继承多个模型, 如下所示写成一个列表值即可, 然后 **\_name** 比如指明了, 因为有多多个继承模型, 不指明 **Odoo** 是不清楚要继承谁的 **\_name** 的。



```
_name = 'todo.task'
_inherit = ['todo.task', 'mail.thread']
```

`mail.thread` 是一个抽象模型，抽象模型没有数据库表达，没有实际创立 SQL 表格。抽象模型最适合被混合继承使用。要创建一个抽象模型就是继承自 `models.AbstractModel` 而不是 `models.Model`。

## 8.6 修改其他数据文件

不像视图文件的 `arch` 结构下的 `xml` 可以用 XPath 表达式，其他 `xml` 数据文件则要采取不同的方法来修改之。

### 8.6.1 删除记录

这是删除记录的语法

```
<delete model="ir.rule" search="[(('id', '=',
    ref('todo_app.todo_task_user_rule')))]" />
```

使用的是 `delete` 标签，然后模型对应某个 `recordset`，然后使用 `search` 方法，这里的 `ref` 语句还不太清楚。

### 8.6.2 更新数据

其他记录若不像删除，则使用 `<record id="x" model="y">` 这样的语法，若该记录不存在，则会创建，若存在则会修改其中的某些值。

下面这个例子是用来修改 `record rule` 权限文件的，将其改成本人和 `follower` 都可以看你的 `todo task`。

```
<?xml version="1.0" encoding="utf-8"?>
<openerp>
  <data noupdate="0">

    <delete model="ir.rule" search="[(('id', '=',
      ref('todo_app.todo_task_user_rule')))]" />
```

```

<record id="todo_task_per_user_rule" model="ir.rule">
  <field name="name">ToDo Tasks only for owner</field>
  <field name="model_id" ref="model_todo_task"/>
  <field name="groups" eval="[(4, ref('base.group_user'))]" />
  <field name="domain_force">
    ['|', ('user_id', 'in', [user.id, False]),
      ('message_follower_ids', 'in', [user.partner_id.id])]
  </field>
</record>

</data>
</openerp>

```

这里的一些细节我们可以先暂时略过，记住这种记录数据删除和更新的方法就是了。然后看到 **data** 标签的 **noupdate** 属性，如果设置为“0”的话就更新数据，这通常是在开发期这样设置，如果在运行期则设置为“1”，也就是接下来模块升级也不会更新本 **data** 数据，通常为了运行期稳定会这样设置。

## 8.7 委托继承

除了前面谈论的 **\_inherit** 继承外，Odoo 还提供了一种继承机制，叫做委托继承 (delegation inheritance)。委托继承特别适合继承官方内置的现有模型。按照官方文档的说法，委托继承一些值是存放于不同的 SQL 表格中的，所以其似乎是通过一种 SQL 连接机制来达到继承效果的。然后委托继承只有 **fields** 被继承了，而方法没有被继承（因为那些方法又不是存放在 SQL 表格里面的。）。

具体委托继承的详情分析还需要进一步讨论。

## 理解模型内的数据文件

## 9.1 理解外部 id

所有的记录在 Odoo 数据库中都有一个独一无二的标识码 `id`, Odoo 是通过 `ir.model.data` 模型来管理这些外部 `id` 的。`ir.model.data` 模型对应的 SQL 表格是 `ir_model_data`。这个表格里面存储着各个模型外部名字 `ID` (通过 `record` 标签的 `id` 属性指定) 和具体数据库某个表格 `ID` 的映射关系。这个表格有四个字段值得引起我们的注意:

我们执行:

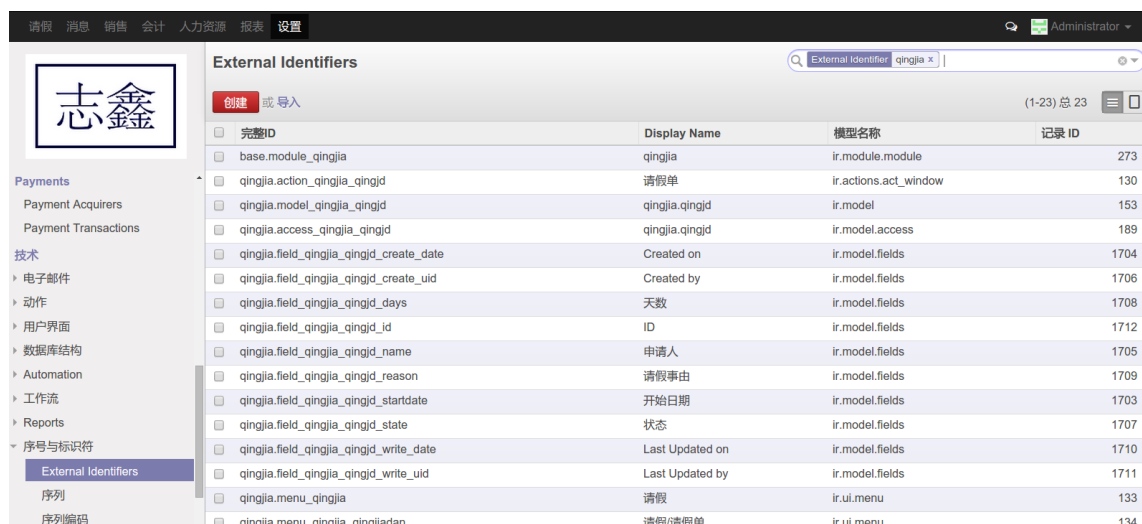
```
SELECT
    id, name, module, model, res_id
FROM
    public.ir_model_data
WHERE
    MODULE = 'qingjia'
;
```

注意 `WHERE` 字句后面的字段要大写。则有:

id	name	module	model	res_id
3707	model_qingjia_qingjd	qingjia	ir.model	153
3708	field_qingjia_qingjd_startdate	qingjia	ir.model.fields	1703
3709	field_qingjia_qingjd_create_date	qingjia	ir.model.fields	1704

3710	field_qingjia_qingjd_name	qingjia	ir.model.fields	1705
3711	field_qingjia_qingjd_create_uid	qingjia	ir.model.fields	1706
3712	field_qingjia_qingjd_state	qingjia	ir.model.fields	1707
3713	field_qingjia_qingjd_days	qingjia	ir.model.fields	1708
3714	field_qingjia_qingjd_reason	qingjia	ir.model.fields	1709
3715	field_qingjia_qingjd_write_date	qingjia	ir.model.fields	1710
3716	field_qingjia_qingjd_write_uid	qingjia	ir.model.fields	1711
3717	field_qingjia_qingjd_id	qingjia	ir.model.fields	1712
3718	access_qingjia_qingjd	qingjia	ir.model.access	189
3719	action_qingjia_qingjd	qingjia	ir.actions.act_window	130
3720	qingjia_qingjd_form	qingjia	ir.ui.view	298
3721	qingjia_qingjd_tree	qingjia	ir.ui.view	299
3722	menu_qingjia	qingjia	ir.ui.menu	133
3723	menu_qingjia_qingjiadan	qingjia	ir.ui.menu	134
3724	menu_qingjia_qingjiadan_qingjiadan	qingjia	ir.ui.menu	135
3725	wkf_qingjia	qingjia	workflow	1
3726	act_draft	qingjia	workflow.activity	1
3727	act_confirm	qingjia	workflow.activity	2
3728	act_accept	qingjia	workflow.activity	3
3729	act_reject	qingjia	workflow.activity	4
3731	qingjia_draft2confirm	qingjia	workflow.transition	1
3732	qingjia_confirm2accept	qingjia	workflow.transition	2
3733	qingjia_confirm2reject	qingjia	workflow.transition	3

然后我们看到



完整ID	Display Name	模型名称	记录 ID
base.module_qingjia	qingjia	ir.module.module	273
qingjia.action_qingjia_qingjd	请假单	ir.actions.act_window	130
qingjia.model_qingjia_qingjd	qingjia.qingjd	ir.model	153
qingjia.access_qingjia_qingjd	qingjia.qingjd	ir.model.access	189
qingjia.field_qingjia_qingjd_create_date	Created on	ir.model.fields	1704
qingjia.field_qingjia_qingjd_create_uid	Created by	ir.model.fields	1706
qingjia.field_qingjia_qingjd_days	天数	ir.model.fields	1708
qingjia.field_qingjia_qingjd_id	ID	ir.model.fields	1712
qingjia.field_qingjia_qingjd_name	申请人	ir.model.fields	1705
qingjia.field_qingjia_qingjd_reason	请假事由	ir.model.fields	1709
qingjia.field_qingjia_qingjd_startdate	开始日期	ir.model.fields	1703
qingjia.field_qingjia_qingjd_state	状态	ir.model.fields	1707
qingjia.field_qingjia_qingjd_write_date	Last Updated on	ir.model.fields	1710
qingjia.field_qingjia_qingjd_write_uid	Last Updated by	ir.model.fields	1711
qingjia.menu_qingjia	请假	ir.ui.menu	133
qingjia.menu_qingjia_qingjiadan	请假/请假单	ir.ui.menu	134

图 9.1: 记录的外部 id

这里的完整 ID 就对应具体的那条记录，其是由 `module` 和 `name` 这两个字段的值组合而成的，比如说 `qingjia.menu_qingjia`，具体格式就是 `<module>.name`。然后具体的内部引用对应的是 `ir_ui_menu` 这个 SQL 表格（根据上面的 `model ir.ui.menu` 而来）中的 133 号记录（根据 `res_id`）而来。

## 9.2 使用外部 id

在 Odoo 新的 API 下，你可以通过这样 `self.env.ref('external id')` 的简介语法来通过外部 id 来引用具体的某个 record。

## 9.3 导出或导入数据文件

在 `tree` 列表视图下，有具体的导入或到处数据文件功能。导入需要 `csv` 格式，导出可以是 `csv` 格式或 `excel` 格式。

值得一提的是 `security` 文件夹下的 `ir.model.access.csv` 文件名字是固定的，然后其他一些访问权限规则最好是单独用文件编写。

然后视图的 `xml` 文件讲起来也是官方内置模块的对象数据文件，不过这里是不能在网页下点击操作的，必须手工编写 `xml` 文件来完成。

`workflow` 的 `xml` 文件推荐放在 `workflow` 文件夹下。

在一定要手工编写 `XML` 文件的情况下，前面已经有所讨论了，这里进一步进行一些补充说明。

## 9.4 快捷输入标签

一般的记录声明就是使用的 `record` 标签，然后加上 `id` 属性和 `model` 属性。如下所示：

```
<record id="group_purchase_user" model="res.groups">
    <field name="name">User</field>
    <field name="implied_ids" eval="[(4, ref('base.group_user'))]" />
    <field name="category_id" ref="base.module_category_purchase_management" />
</record>
```

同时前面提到了 `menuitem` 这样的快捷输入标签可以这样使用。

```
<menuitem id="menu_qingjia" name=" 请假 " sequence="0"></menuitem>
```

这些快捷标签的使用很方便的，下面是一些可用的快捷输入标签清单：

`<act_window>` 对应模型 `ir.actions.act_window`，视窗动作对象。

`<menuitem>` 对应模型 `ir.ui.menu`，菜单对象。

`<report>` 对应模型 `ir.actions.report.xml` 打印动作对象。

`<template>` 对应模型 `ir.ui.view`，视图的模板文件对象。

`<url>` 对应模型 `ir.actions.act_url` URL 打开动作对象。

其他的就好用 `record` 标签的标准形式来引入对象数据记录了。

## 9.5 用 field 标签设置值

具体指定某个字段的值如上使用 `field` 标签，然后用 `name` 指明某个字段。

具体的值字符串不需要加上双引号""，直接写上即可。布尔值直接写上 `0`，`1` 或者 `False`，`True` 都是可以的。返回日期或日期时间采用如下格式也可以正确被转换：`YYYY-MM-DD` 和 `YYYY-MM-DD HH:MI:SS`。

### 9.5.1 eval 语法

前面也有所涉及，`field` 的值支持用 `eval` 语法来运算某个表达式获得。如下所示，其内任意的 `python` 表达式都是可以的：

```
<field name="expiration_date"
    eval="(datetime.now() + timedelta(-1)).strftime('%Y-%m-%d')" />
```

`datetime` 模块下的 `datetime` 还有 `timedelta` 类已经被引入进来可以直接使用了。`datetime` 模块的介绍不是这里的重点，所以这里的细节略过了。

然后 `ref` 函数也可以直接使用：

```
<field name="user_id" eval="ref('base.group_user')" />
```

上面使用 `ref` 函数引用记录外部 `id base.group_user`，其是 `res_groups` 表格的第五条记录，具体在群组里面是 `employee`（人力资源/雇员）组。

```
<value model="sale.order"
    eval="obj(ref('test_order_1')).amount_total" />
```

这里的 `obj` 是根据某个记录来得知具体的某个模型（还不清楚？？）

### 9.5.2 ref 属性

`ref` 函数在这里对应的 `field` 主要是 `Many2One` 类型的 `field`。不过更简单的可以不用 `eval` 而直接用 `ref` 属性来调用。比如上面的就可以简单写为：

```
<field name="user_id" ref="base.group_user" />
```

### 9.5.3 One2many 和 Many2many 的 eval 赋值

```
<field name="tag_ids"
    eval="[(6,0,
        [ref('vehicle_tag_leasing'),
         ref('fleet.vehicle_tag_compact'),
         ref('fleet.vehicle_tag_senior')]
    )]" />
```

- `(0,_,{'field': value})` 这将创建一个新的记录并连接它
- `(1,id,{'field': value})`：这是更新一个已经连接了的记录的值

- (2,id,\_) 这是删除或取消连接某个已经连接了的记录
- (3,id,\_) 这是取消连接但不删除一个已经连接了的记录
- (4,id,\_) 连接一个已经存在的记录
- (5,\_,\_) 取消连接但不删除所有已经连接了的记录
- (6,\_[ids]) 用给出的列表替换掉已经连接了的记录

这里的下划线一般是 0 或 False。



## ODOO 开发基础：请假模块第二谈

`__init__.py` 文件没啥好改动的，然后我们再看到 `main_model.py` 文件，这一次进行了较多地方的改动。

```
from openerp import models, fields, api
import logging

class Qingjd(models.Model):
    _name = 'qingjia.qingjd'

    name = fields.Many2one('hr.employee', string=" 申请人", readonly=True)

    manager = fields.Many2one('hr.employee', string=" 主管",readonly=True)

    beginning = fields.Datetime(string=" 开始时间", required=True,
        default = fields.Datetime.now())
    ending = fields.Datetime(string=" 结束时间", required=True)
    reason = fields.Text(string=" 请假事由",required=True)

    accept_reason = fields.Text(string=" 同意理由",default=" 同意。")

    #####compute 没有写入数据库 on the fly 可以被 workflow 的 condition 调用
    current_name = fields.Many2one('hr.employee', string=" 当前登录人",compute="_get_current_name")
```

```

is_manager = fields.Boolean(compute='_get_is_manager')

#####

state = fields.Selection([
    ('draft', " 草稿"),
    ('confirmed', ' 待审核'),
    ('accepted', ' 批准'),
    ('rejected', ' 拒绝'),
],string=' 状态',default='draft',readonly=True)

@api.model# 使用新的 api
def _get_default_name(self):
    uid = self.env.uid
    res = self.env['resource.resource'].search([('user_id','=',uid)])
    name = res.name
    employee = self.env['hr.employee'].search(
        [('name_related','=',name)])

    # for i in self.env.user:# 说明其是 recordset
    #     print('hello')

    return employee

@api.model
def _get_default_manager(self):# 单记录 recordset 可以直接用点记号读取属性值
    uid = self.env.uid
    res = self.env['resource.resource'].search([('user_id','=',uid)])
    name = res.name
    employee = self.env['hr.employee'].search(
        [('name_related','=',name)])
    logging.info("myinfo {}".format(employee.parent_id))

    return employee.parent_id # 似乎有这种数字引用方法值得我们注意

```

```

_defaults = {
    'name' : _get_default_name ,
    'manager' : _get_default_manager ,
}

def _get_is_manager(self):### 这里 return 不起作用
    print('-----test')
    print(self.current_name, self.manager, self.env.uid)
    if self.current_name == self.manager:
        self.is_manager = True
    else:
        self.is_manager = False

def _get_current_name(self):
    uid = self.env.uid
    res = self.env['resource.resource'].search([('user_id','=',uid)])
    name = res.name
    employee = self.env['hr.employee'].search(
        [('name_related','=',name)])

    self.current_name = employee

#####

def draft(self, cr, uid, ids, context=None):
    if context is None:
        context={}
    self.write(cr,uid,ids,{'state':'draft'},context=context)
    return True

def confirm(self, cr, uid, ids, context=None):
    if context is None:
        context={}
    self.write(cr,uid,ids,{'state':'confirmed'},context=context)
    return True

```

```

def accept(self, cr, uid, ids, context=None):
    if context is None:
        context={}
    self.write(cr,uid,ids,{'state':'accepted'},context=context)
    print(' 你的请假单被批准了')
    return True

def reject(self, cr, uid, ids, context=None):
    if context is None:
        context={}
    self.write(cr,uid,ids,{'state':'rejected'},context=context)
    print(' 抱歉, 你的请假单没有被批准。')
    return True

```

读者不要着急，这里的内容我会慢慢讲解的。然后 `views.xml` 改成了这个样子：

```

<?xml version="1.0"?>
<openerp>
<data>

<!--
    打开请假单动作
-->

<act_window id="action_qingjia_qingjd"
    name=" 请假单"
    res_model="qingjia.qingjd"
    view_mode="tree,form" />

<!--
    表单视图
-->

<record id="qingjia_qingjd_form" model="ir.ui.view">
<field name="name">qing jia dan form</field>
<field name="model">qingjia.qingjd</field>
<field name="arch" type="xml">
    <form>

```

```

<header>
    <button name="btn_confirm" type="workflow" states="draft"
    string=" 发送" class="oe_highlight" />
    <button name="btn_accept" type="workflow" states="confirmed"
    string=" 批准" class="oe_highlight"/>
    <button name="btn_reject" type="workflow" states="confirmed"
    string=" 拒绝" class="oe_highlight"/>
    <field name="state" widget="statusbar" statusbar_visible="draft,confirmed,accepted,r
</header>

<sheet>
    <group name="group_top" string=" 请假单">
        <group name="group_left">
            <field name="name"/>
            <field name="beginning"/>
        </group>
        <group name="group_right">
            <field name="manager"/>
            <field name="ending"/>
        </group>
    </group>
    <group name="group_below">
        <field name="reason"/>
    </group>
</sheet>

</form>
</field>
</record>
<!--
tree 视图
-->

<record id="qingjia_qingjd_tree" model="ir.ui.view">
    <field name="name">qing jia dan tree</field>
    <field name="model">qingjia.qingjd</field>

```

```

<field name="arch" type="xml">
    <tree>
        <field name="name"/>
        <field name="beginning"/>
        <field name="ending"/>
        <field name="state"/>
    </tree>
</field>
</record>
<!--
加入菜单
-->
<menuitem id="menu_qingjia" name=" 请假" sequence="0"></menuitem>
<menuitem id="menu_qingjia_qingjiadan" name=" 请假单" parent="menu_qingjia"></menuitem>
<menuitem id="menu_qingjia_qingjiadan_qingjiadan" parent="menu_qingjia_qingjiadan" action="a
</data>
</openerp>

```

除了跟着 `main_model.py` 文件里面的一些修改而更改外，最值得一提的就是 `button` 的 `type` 属性设置为了“`workflow`”。如果还是设置为“`object`”，当你点击按钮的时候，该模型就应该提供对应按钮 `name` 的一个方法，这个方法将执行某些动作。这里要强调的就是这样简单的 `object` 按钮是不和后面谈到的工作流的概念兼容的。如果按钮的 `type` 还是设置为 `object`，那么其是不发送工作流的 `signal` 的。更多工作流的细节等下再谈。

然后还编写了一个工作流的 `workflow.xml` 文件：

```

<?xml version="1.0" ?>
<openerp>
<data noupdate="0">
    <record id="wkf_qingjia" model="workflow" >
        <field name="name">wkf.qingjia</field>
        <field name="osv">qingjia.qingjd</field>
        <field name="on_create">True</field>
    </record>

    <record id="act_draft" model="workflow.activity" >
        <field name="wkf_id" ref="wkf_qingjia" />

```

```

    <field name="name">draft</field>
    <field name="flow_start" eval="True"/>
    <field name="kind">function</field>
    <field name="action">draft()</field>
</record>

<record id="act_confirm" model="workflow.activity" >
    <field name="wkf_id" ref="wkf_qingjia" />
    <field name="name">confirm</field>
    <field name="kind">function</field>
    <field name="action">confirm()</field>
</record>

<record id="act_accept" model="workflow.activity">
    <field name="wkf_id" ref="wkf_qingjia" />
    <field name="name">accept</field>
    <field name="kind">function</field>
    <field name="flow_stop">True</field>
    <field name="action">accept()</field>
</record>

<record id="act_reject" model="workflow.activity" >
    <field name="wkf_id" ref="wkf_qingjia" />
    <field name="name">reject</field>
    <field name="kind">function</field>
    <field name="action">reject()</field>
</record>

<record model="workflow.transition" id="qingjia_draft2confirm">
    <field name="act_from" ref="act_draft" />
    <field name="act_to" ref="act_confirm" />
    <field name="signal">btn_confirm</field>
</record>

```

```

<record model="workflow.transition" id="qingjia_confirm2accept">
    <field name="act_from" ref="act_confirm" />
    <field name="act_to" ref="act_accept" />
    <field name="signal">btn_accept</field>
    <field name="condition">is_manager</field>
</record>

<record model="workflow.transition" id="qingjia_confirm2reject">
    <field name="act_from" ref="act_confirm" />
    <field name="act_to" ref="act_reject" />
    <field name="signal">btn_reject</field>
    <field name="condition">is_manager</field>
</record>

</data>
</openerp>

```

然后 `__openerp__.py` 文件没做什么修改, 就把上面的 `workflow.xml` 加进来, 然后 `depends` 将 `base` 改为了 `hr`, 因为等下一些功能是依赖官方内置模块 `hr` (人资管理模块) 的。

读者可以按照上面的描述先看看这个请假模块第二版是什么样子, 下面是一个简单的示意图:

图 10.1: 请假模块第二版



这么一张简单的图并没很好地第二版的加强功能说明出来，建议读者新建几个用户，然后在人力资源那里把员工和部门设置好，尤其是部门主管等信息。然后读者可以尝试以一个普通员工来新建一个请假单，编辑，保存，发送。然后接下来的批准和拒绝按钮该普通员工是不能点击的（工作流的 **condition** 控制的），而只有主管才可以正常点击批准或拒绝按钮。

## 10.1 本例涉及到的数据库表格简介

这里我只进行简略的讨论，具体该表格的细节请读者自己用 **pgadmin3** 软件查看之。

**res.users** 在网页视图下对应的菜单是：设置 → 用户 → 用户。这个表格（或说模型）存储着一些登录用户的信息，比如用户名或密码等。

**res.groups** 在网页视图下对应的菜单是：设置 → 用户 → 组。权限管理就是利用了这个表格建立的群组概念，比如最常使用的群组，人力资源/雇员就是这个表格的第五条记录，其外部 **id** 是 **base.group\_user**。

**resource.resource** **res.users** 存储的只是用户的登录名和密码等，而在网页上具体显示的是该用户更人类易读的名字，其就是根据这个表格来完成的。这个表格有 **user\_id**，该 **id** 就对应 **res.users** 的某条记录，而 **name** 就是对应的人类易读的名字。

**hr.employee** 在网页视图下对应的菜单是：人力资源 → 人力资源 → 员工。这个表格存储着员工的一些信息，其中 **name\_related** 属性就对应前面 **resource.resource** 的 **name**，然后 **parent\_id** 对应该员工的上一级也就是主管，具体就是本表格的对应 **id** 记录，然后 **department** 是该员工所属部门，具体是 **hr.department** 表格中的对应 **id** 的记录。

**hr.department** 在网页视图下对应的菜单是：人力资源 → 设置 → 部门。本表格记录了公司的部门信息（支持多公司概念，有 **company\_id** 标识），还有该部门的主管对应的员工 **id**。

## 10.2 工作流概念入门

工作流对象算是 **Odoo** 框架里面颇具特色的一部分，其在网页视图中对应的菜单是：设置 → 工作流 → 工作流。对应的模型是 **workflow**，但并没有 **workflow** 这个表格，要说对应的应该是 **wkf** 这个表格，然后还有 **wkf\_activity** 表格等。

**Odoo** 工作流的图标视图如下图所示是一个不错的查看和管理当前工作流程的工具：

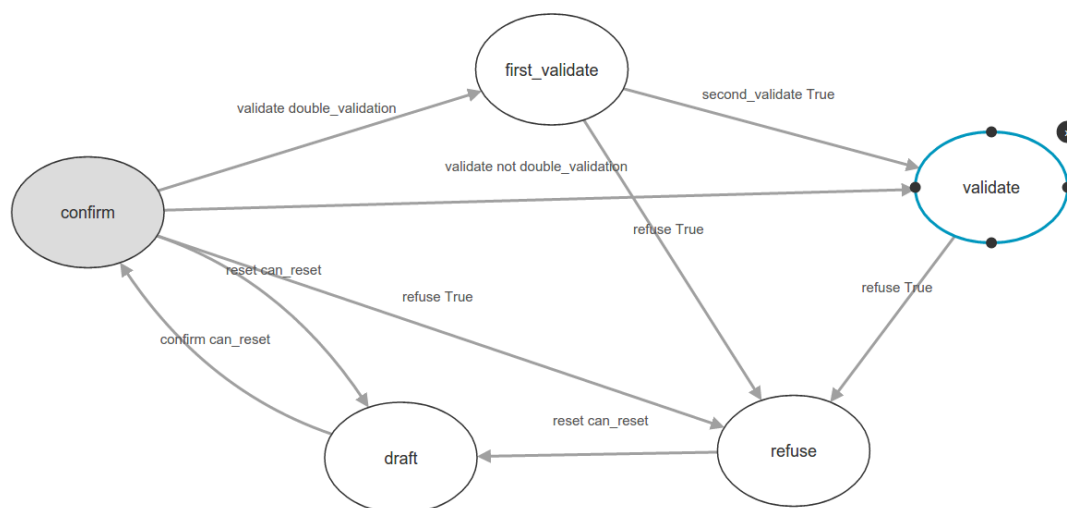


图 10.2: 工作流图表视图示意图

其中节点叫做“活动 (activity)”，然后弧线连接叫做“转变 (transition)”。活动描述了 Odoo 服务器应该完成的一些工作，比如：改变某些记录的状态，发送 email 等等。转变则控制了从活动到活动之间的工作流。

转变可以增加属性如条件、信号、触发器等。这样工作流的行为是取决于用户的动作（如点击某个按钮），或某个记录值的更改或任意的 python 代码。总而言之，Odoo 工作流系统提供了：

- 关于记录如何演变的描述
- 根据多样的弹性的条件建立自动化行动机制
- 管理公司流程和确认规则
- 管理对象间的互动
- 在他们的生命期内一个可视化的流程图

工作流和模型关联在一起，具体某个模型新建一个对象就会是实例化一个工作流对象。然后该工作流对象应该有个状态记录，服务器重启之后工作流还是在那个状态下，也就是已经执行了的都不会执行了。

### 10.2.1 定义工作流对象

```

<record id="wkf_qingjia" model="workflow" >
    <field name="name">wkf.qingjia</field>
    <field name="osv">qingjia.qingjd</field>

```

```
<field name="on_create">True</field>
</record>
```

首先是定义了本工作流对象，**model=workflow**，**id** 是多少，这个 **id** 很重要，等下本工作流对象里面的活动和转变都要用到这个 **id**。

然后 **field** 定义了

**name** 本工作流的 **name**，随意。

**osv** 这个属性很重要，是具体的那个模型，其和本工作流关联起来了。

**on\_create** 一般设置为 **True**，工作流会根据每一个模型新建一个对象再实例化一次。

### 10.2.2 创建节点

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

这里创建了一个节点活动对象：

**flow\_start** 如果设置为 **True** 工作流会从这里开始

**wkf\_id** 属于某个工作流对象我们看到这里的 **ref** 语法如果是引用本模型内的对象则可以省略。

**kind** 有四种 **dummy function subflow stop all**

**action** 具体的动作（于模型上）

**flow\_stop** 一个工作流的完成就是所有的活动有 **flow\_stop** 属性的都设置为了 **True**

### 10.2.3 创建连接

```
<record model="workflow.transition" id="holiday_draft2confirm"> <!-- 1. draft->submitted (confirm) -->
  <field name="act_from" ref="act_draft" />
```

```
<field name="act_to" ref="act_confirm" />
<field name="signal">confirm</field>
<field name="condition">can_reset</field>
<field name="group_id" ref="base.group_user"/>
</record>
```

转变对象属于 `workflow.transition` 模型。

`act_from` 何活动出发 `act_to` 到何活动去 `signal`

`condition`

2.1.2 Accessing Current User `self.env.user`

`'lang' ('self.env.lang')`.

`default` 使用 `lambda` 语句其默认接受 `self` 参数, 即本模型具体的对象。

`ir.module.category`

## ODOO 模型层详解

经过前面的介绍，我们也确实感觉到 Odoo 的 ORM 层的 API 应该是 Odoo 技术框架最核心的部分，如果我们翻翻 Odoo 框架的源码，也会看到 `models.py` 那个文件有六七千行的代码，这也说明 Odoo 的设计者在编写 ORM 这块是花费了很多精力的。所以我想我们把 Odoo 的 ORM 层的 API 这部分知识掌握了，Odoo 框架的神秘面纱也基本上被掀开一大半了。本章将在前面讨论的基础上进一步详细介绍 Odoo ORM API 的细节。

### 11.1 `_name`

定义了本模型具体对应 SQL 表格的名字，比如前面定义的 `mymodule.fruits` 对应的数据库中的表格名是 `public.mymodule_fruits`。

### 11.2 各个表头属性

```
class Fruits(models.Model):  
    _name = 'mymodule.fruits'  
  
    name = fields.Char()
```

如上所示这样定义的类的一个 `name` 属性就对应 SQL 中的一个表头名，即名叫 `name` 的一列。这个我们在前面的 `pgadmin3` 中看到了的。

然后上面的 `fields.Char()` 具体定义了一个字符串输入字段，类似的还有 `fields.Boolean` 布尔值；`fields.Integer` 整数值；`fields.Float` 浮点数值；`fields.Text` 和 `Char` 类似，但通常用于多行文本字段输入；`fields.Selection` 几个值的选择；`fields.Html`；`fields.Date`；`fields.Datetime` 等。这些都是所谓的简单字段输入，此外还有一种关系字段，其是用于描述表格之间的关系的（相同模型或者不同模型）。

`fields.Char()` 函数可以接受一些可选参数，比如 `string` 表示本模型为用户可见的名字；`required` 接受一个布尔值，默认是 `False`，如果是 `True`，则该字段不可为空值，其要某有个默认值要某有个设定值；`help` 在用户 UI 界面下的帮助信息；`index` 布尔值，默认是 `False`，如果为 `True` 则要求在数据库中为这列创建一个索引 (`index`)。

然后我们在 `pgadmin3` 前面的介绍中也看到了，此外还有创建一些其他的表头字段：

- `id` 在表格中一条记录的独特 `id`
- `create_date` 创建日期
- `create_uid` 谁创建的
- `write_date` 最后修改日期
- `write_uid` 谁修改的

## 11.3 name 字段

Odoo 中的模型一般都还需要 `name` 字段，用于各种搜索或显示行为。

## 11.4 具体模型的数据

具体模型的数据是用 XML 文件来声明的，如下所示：

```
<openerp>
  <data>
    <record id="apple" model="mymodule.fruits">
      <field name="name">apple</field>
    </record>
  </data>
</openerp>
```

这里的 **record** 元素你可以理解为 SQL 表格的一条记录，或者 Odoo 模型具体的一个实例一个对象。然后 **id** 属性特别标记了这条记录（说可被外调用，具体还不清楚）；**model** 属性就是这个对象具体对应那个模型。

然后里面的 **field** 元素你可以看作某条记录具体的某个 **name** 表头的字段，**field body** 里面就放着这个字段的值。

这里的 **record** 在视图中对应的是 **basic view**；此外还有 **tree** 对应的是列表视图；此外还有 **form** 对应的是表单视图。

## 11.5 模型间的关系

Many2one

One2many

Many2many

## 11.6 工作流

一个工作流模型在 **Session** 模型上都加入了 **state** 字段：有三种字段，Draft Confirmed Done

有效的转变有：

Draft → Confirmed Confirmed → Draft Confirmed → Done Done → Draft





---

CHAPTER

TWELVE

---

ODOO 视图层详解



## 13.1 Odoo 里老的 API

`_name` 点号记法对应具体 SQL 表名 `_columns {`

`}` 表述 SQL 表头

`_defaults` 字典值描述默认值

`_inherit`

`_inherits` 委托继承

## 13.2 PostgreSQL 数据库命令行操作

### 13.2.1 命令行数据库备份

## 13.3 反向代理 (reverse proxy)

反向代理如下图所示【图片来自 wiki】

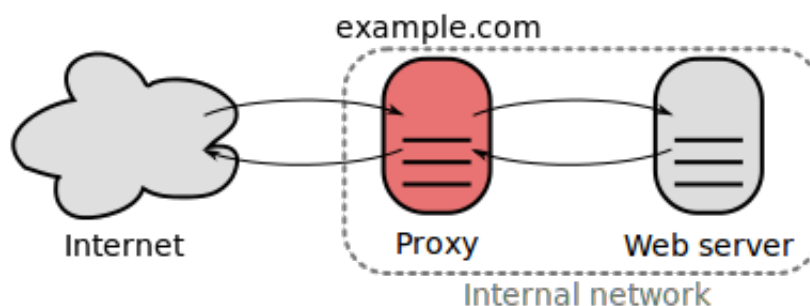


图 13.1: 反向代理

其之所以叫做反向是相对于我们常见的代理翻墙软件而言的，翻墙软件叫做前向代理，其代理是给客户端用的，而反向代理具体是给服务器端用的。反向代理的用途有加密，安全，缓存，压缩负载均衡等等。有名的 **Apache** 软件就是一个反向代理软件，此外还有 **nginx** 软件也是一个不错的反向代理软件。

### 13.3.1 安装 **nginx** 软件

#### 释放 **http** 默认端口号

**http** 的默认端口号是 80，这不应该被占用，可使用 **curl** 命令检测之。

其应该返回连接不上的错误：

```
$ curl http://localhost
curl: (7) Failed to connect to localhost port 80
```

如果有响应，则需要将对应的服务停用掉。然后再安装 **nginx** 软件。

在 **ubuntu** 下安装 **nginx** 软件很简单，使用 **apt-get** 安装即可：

```
sudo apt-get install nginx
```

安装完了之后再 **curl http://localhost** 应该返回 **nginx** 的一些信息了。

然后人们一般把默认的配置文件给删除了：

```
sudo rm /etc/nginx/sites-enabled/default
```

在 **/etc/nginx/available-sites/** 编写可用网站的配置文件，如果想具体使用该网站的配置文件，就创建一个符号连接到目录 **/etc/nginx/enabled-sites/**。

具体创建一个 **odoo** 的配置文件如下：

```
sudo touch /etc/nginx/sites-available/odoo
sudo ln -s /etc/nginx/sites-available/odoo /etc/nginx/sites-enabled/odoo
```

然后开始编辑配置 `available-sites` 文件夹下的那个 `odoo` 文件。初步将文件修改如下：

```
upstream backend-odoo {
    server 127.0.0.1:8069;
}

server {
    location / {
        proxy_pass http://backend-odoo;
    }
}
```

这里 `upstream` 可看作上传流，名字，然后具体对应的端口号。然后服务器 `server` 的 `location` 应该是入口点，然后 `proxy_pass` 这里的 `http://backend-odoo` 应该就具体对应前面 `upstream` 的那个 `backend-odoo`。具体细节后面再慢慢了解。

然后执行：

```
sudo nginx -t
```

来测试下 `nginx` 是否工作正常，正常的话再重启下 `nginx` 服务。

```
sudo /etc/init.d/nginx reload
```

然后进入浏览器输入 `localhost`，我们已经能够正常登入 `odoo` 了。

### 13.3.2 强制 https 连接

### 13.3.3 nginx 优化

### 13.3.4 轮询机制

## 13.4 跟踪项目源码初始化进程

首先我们看到 `odoo.py` 的 `main` 函数，正常的输入 `./odoo.py` 前面的 `if` 语句都没有执行，直接跳转到 `openerp.cli.main()` 那里。

然后接下来执行一个 `openerp.cli.server.Server` 对象，并调用这个对象的 `run` 方法，继而执行 `cli` 文件夹下 `server.py` 的 `main` 函数，然后 `args` 参数也传递过来了（比如 `['--addons-path=addons,myaddons']`）。

接下来执行 `check_root_user` 函数，检查当前用户是不是 `root` 用户，如果是这样很不安全，程序终止。

首先是初始化这个对象，在 `openerp.tools` 的 `config.py` 文件中，初始化 `class configmanager`。一些基本的参数设置和命令行选项帮助信息等的设置。接下来处理传递过来的参数 `openerp.tools.config.parse_config(args)`，实际的刷参数和设置参数。

然后是 `check_postgres_user` 检查数据库用户是不是 `postgres`，如果是太过于危险，程序退出。

然后是 `report_configuration` 报告配置情况。

配置初始化之后，根据配置进行了一些基本的抉择，然后执行 `setup_pid_file` 函数，为进程创建一个文件，还不太懂。

然后是这个：

```
rc = openerp.service.server.start(preload=preload, stop=stop)
```

具体是 `openerp` 的 `service` 文件夹的 `server.py` 的 `start` 函数，参数我们假设这里都是默认值 `None`，这个函数就是启动 `openerp` 的 `http` 服务器和 `cron` 处理器。

`server` 这个服务器变量被定义为了一个全局变量，然后开始执行 `load_server_wide_modules` 这个函数，

然后接下来是：

```
INFO:root:now i am load the module web
```

```
INFO:root:now i am load the module web_kanban
```

通过调用 `openerp→modules` 文件夹下的 `module.py` 的 `load_openerp_module` 函数来加载某个模块。

这是避免重复加载:

```
global loaded
if module_name in loaded:
    return
```

接下来就是具体处理模块的路径问题, 这里有些细节先略过, 然后就是将其作为一个 `python` 模块加载进来, 然后读取模块的基本信息。

作为 `python` 模块大家都清楚必须要有 `__init__.py` 文件, 然后模块的基本信息是写在 `__openerp__.py` 文件里面的。然后模块那边还有很多细节问题, 比如 `__init__.py` 里面一般有

```
import controllers
```

也就是本文件夹下应该有个 `controllers.py` 文件, 其内代码将被加载或说被执行。然后 `__openerp__.py` 里面描述了包的依赖, 从其中我们可以看到, 它们都依赖于 `openerp→addons` 里面的 `base` 模块, 这个 `base` 模块里面的细节也需要详细讨论。这些细节在这里我们先略过。现在假设 `web` 和 `web_kanban` 这两个模块都加载好了。

```
if openerp.evented:
    server = GeventServer(openerp.service.wsgi_server.application)
elif config['workers']:
    server = PreforkServer(openerp.service.wsgi_server.application)
else:
    server = ThreadedServer(openerp.service.wsgi_server.application)
```

我们再看到 `openerp→service` 的 `wsgi_server.py` 文件的 `application` 函数。

假设没设置代理的话, 其对应的是 `application_unproxied(envIRON, start_response)` 这就是 `WSGI` 接口层的一些东西, 这里的细节先略过, 其中还进行了线程封装等等, 然后就是:

```
rc = server.run(preload, stop)
```

就是具体开始运行服务器了。至此程序进入和用户交互的环节了。

在服务器进入和用户交互之后，如果你输入 `127.0.0.1:8069`，程序要处理两个 handler:

```
2015-05-11 08:40:16,875 6807 INFO ? root: i found the handler <function wsgi_xmlrpc at 0xb56d261
2015-05-11 08:40:16,876 6807 INFO ? root: i found the handler <openerp.http.Root object at 0xb54
```

第一个是 `wsgi_xmlrpc`，第二个就是 `openerp.http.Root` 的一个实例。其中第一个我估计和具体 `xml rpc` 通信有关，然后第二个就是具体刷新根目录需要做的工作。然后我们看到

```
2015-05-11 08:40:17,011 6807 INFO ? openerp.addons.bus.bus: Bus.loop listen imbus on db postgres
2015-05-11 08:40:17,246 6807 INFO ? openerp.addons.report.models.report: You need Wkhtmltopdf to
2015-05-11 08:40:17,329 6807 INFO ? openerp.http: HTTP Configuring static files
2015-05-11 08:40:17,336 6807 INFO odoo openerp.modules.loading: loading 1 modules...
2015-05-11 08:40:17,344 6807 INFO odoo openerp.modules.loading: 1 modules loaded in 0.01s, 0 que
2015-05-11 08:40:17,361 6807 INFO odoo openerp.modules.loading: loading 47 modules...
2015-05-11 08:40:17,443 6807 INFO odoo openerp.modules.loading: 47 modules loaded in 0.08s, 0 qu
2015-05-11 08:40:18,040 6807 INFO odoo openerp.modules.loading: Modules loaded.
2015-05-11 08:40:18,041 6807 INFO odoo openerp.addons.base.ir.ir_http: Generating routing map
2015-05-11 08:40:18,643 6807 INFO odoo werkzeug: 127.0.0.1 - - [11/May/2015 08:40:18] "GET / HTTP
```

程序一是建立和数据库的连接，然后进行 `HTTP` 的相关配置，然后加载模块，然后是生成寻址 `map`，然后就是通过 `werkzeug` 进行 `WSGI` 通信，`GET /`。后面的内容就会更加琐碎了，就是具体什么路径什么模块，什么 `WSGI` 通信和响应等等了。这一块我们先暂时略过，先看看 `web` 和 `web_kanban` 这两个模块。这两个模块在服务器还没有启动之前就加载了，所以是些什么内容呢？然后还有 `base` 模块肯定也已经加载进去了。

### 13.4.1 base 模块

因为其他所有的模块都依赖于 `base` 模块，所以 `base` 模块的一些内容会被其他模块调用，比如在 `web` 的 `controllers` 下的 `main.py` 中就有这样的导入语句：

```
from openerp.addons.base.ir.ir_qweb import AssetsBundle, QWebTemplateNotFound
```

这个 `openerp.addons.base.ir` 里面有很多 `py` 文件，要了解起来是要花费一定的时间的，这个 `ir_qweb.py` 看起来和 `QWEB` 模板系统有关。然后还有其他很多内容。。



### 13.4.2 web 模块

随便翻翻 web 模块的 controllers 文件夹下的 main.py 文件，其主要和数据库管理页面还有登录页面有关。

### 13.4.3 web\_kanban 模块

这个模块没什么 python 代码，似乎就是前面 web 模块的一些数据补充？

## 13.5 配置会计科目

### 13.5.1 配置会计科目类型

看到会计 → 设置 → 科目 → 科目类型。

新建下面四项：

类型	编码	P&L/BS 分类	结转方式
视图	视图	/	不结转 (None)
收入	收入	损益表 (收益)[Profit & Loss (Income Accounts)]	结转未核销往来明细 (unreconciled)
支出	支出	损益表 (费用)[Profit & Loss (Expense Accounts)]	结转未核销往来明细 (unreconciled)
现金	现金	资产负债表 (资产)[Balance Sheet (Assets Accounts)]	结转余额 (Balance)

这里的 P&L/BS 分类还有一个 Balance sheet(Liability account) 中文翻译是负债和权益类，觉得应该翻译为资产负债表 (负债)，以后核实之后再说。

然后结转方式，参考了 [这个网页](#)，中文翻译一眼看过去估计只有那些专业人员才看得懂。

1. **None** 原翻译是不结转，我觉得应该翻译为无，意思是年末结转的时候不会考虑这个类型的科目。
2. **balance** 翻译为结转余额，还行吧，意思是年末结转时会把这个科目的余额转为下年的期初余额。
3. **details** 翻译为结转所有往来明细，还行，。意思是年末结转时会把该科目的凭证行逐行转入下年。
4. **unreconciled** 翻译为结转未核销往来明细，意思是年末结帐时会把该科目未对账的凭证行逐行转入下年。可简单翻译为未对账。

### 13.5.2 配置会计科目

看到会计 → 设置 → 科目 → 科目。

需要会计科目来处理未付款的销售订单和采购订单，货物的收发两个以上，资金的收付款一个。还有一个组织科目视图。所以需要如下 6 个会计科目。

名称	code	内部类型	上级科目	账户类型	对账
最小的图	0	视图		视图	未选中
应付	AP	应付	最小的图	应付	已检查
应收	AR	应收	最小的图	应收	已检查
现金	C	流动资金	最小的图	现金	未选中
采购	P	常规科目	最小的图	支出	未选中
销售	S	常规科目	最小的图	收入	未选中

## 13.6 分录

这个分录也就是所谓的 Journals 日记帐。在会计 → 设置 → 分录 → 分录那里。

新建下面三个条目：

日记帐名称	code	类型	对象实体序列	默认借方科目	默认贷方科目
采购日记帐	PUJ	采购	采购日记帐 (Purchase Journal)	P 购买	P 购买
销售日记帐	SAJ	销售	账户默认销售日志 (Account Default Sales Journal)	S 卖出	S 卖出
银行日记帐	BNK	现金	账户日记 (Account Journal)	C 现金	C 现金

## 13.7 新建业务伙伴

### 13.7.1 新建业务伙伴标签

在销售 → 设置 → 地址簿 → 合作伙伴标签那里。

新建两个业务伙伴标签：供应商和客户。

### 13.7.2 新建客户

在销售 → 销售 → 客户那里。

点击创建来新建一个客户：

## 13.8 创建新的产品

在仓库 → 产品 → 产品那里或者销售 → 产品 → 产品那里可以创建新的产品。

在需求 → cost price 那里输入成本价；在标价那里输入销售价。

## 13.9 设置会计年度

在会计 → 设置 → 会计期间 → 会计年度那里。

设置好一年之后可以点击创建季度从而自动生成几个季度的会计期间。

## 13.10 向供应商下单

在采购 → 采购 → 询价单那里。

新建一个询价单。

## 13.11 会计学入门

会计系统反映了企业的两个基本方面：自己有什么和欠别人什么。资产 (asset) 是企业拥有或控制的能够给企业带来未来经济效益的资源。例如：现金，物料，设备和土地等。负债 (liabilities) 是企业欠非其所有者（债权人）的债务，未来需要用现金、产品或服务来偿还。权益 (equity) 是指企业所有人对企业资产所享有的求偿权。

总的来说有：

$$\text{资产} = \text{负债} + \text{所有者权益}$$

负债已经放在所有者权益前面，因为负债应该先被满足。此外还有一个扩展的会计公式：

$$\text{资产} = \text{负债} + \text{所有者名下的资本} - \text{所有者提取} + \text{收入} - \text{费用}$$

其中收入减去费用的部分便是净利润 (net income)，如果费用大于收入，则会产生净损失 (net loss)。

常见的经济业务：

- 所有者投资谁成立了一家公司，然后以该公司的名义存了多少钱，创始人的这笔钱叫做所有者投资，属于前面的所有者权益中的所有者名下的资本。
- 用现金采购物料这是将公司的现金资产转变成为另一种资产 (物料)，这项经济业务仅仅改变了资产形式。
- 用现金购买设备这和上面类似，也仅仅改变了资产形式，公司资本总数量还是没变。
- 赊购物料公司资产增加，赊的钱属于公司的负债。
- 提供服务赚取现金资产现金部分增加，右边属于收入增加。
- 用现金支付费用资产现金部分减少，右边费用增加，然后减去。
- 用赊销的方式提供服务或出租设备左边资产应收款项部分增加，右边收入部分增加。
- 应收账款变现左边资产由应收款变为现金，右边没有变化。
- 支付应付款项应付款项在左边属于现金减少，在右边属于负债减少。
- 所有者提取现金在左边资产现金减少，右边所有者权益减少。

### 13.11.1 财务报表

利润表所有者权益表资产负债表现金流量表

### 13.11.2 原始凭证

原始凭证可以是纸质的也可以是电子版的，主要有：销售发票，支票，订货单，供货商签发的账单，员工收入记录，银行对账单等。

### 13.11.3 账户

资产类账户 = 负债类账户 + 所有者权益账户

#### 资产类账户

资产是指企业拥有或控制的预计在未来能够给企业带来一定的经济效益的资源。大多数会计系统都包含以下账户：

- 现金账户 (cash) 反映企业的现金金额，现金的增减变动情况都要记录在现金账户中。
- 应收款项 (account receivable) 是指卖方持有的买房对卖方的付款承诺。

- 应收票据 (note receivable) 也称为期票，是一种书面承诺，承诺在未来某个特定时间还款。
- 预付款项 (prepaid accounts) 代表着提前支付的未来费用。
- 物料 (supplies) 在被使用完之前属于资产，使用完之后成本将被记入费用账户。
- 设备 (equipment) 也是一项资产，随着设备的使用和耗费，其成本将一点点列为费用，这种费用成为折旧。
- 建筑物 (buildings)
- 土地 (land)

### 负债类账户

比较常见的负债类账户有：

- 应付款项 (account payable) 口头的或暗含以后要付款的承诺。
- 应付票据 (note payable) 较为正式的未来付款承诺。
- 预收账款 (unearned revenues) 未来企业提供产品或劳务才能清偿的负债。
- 应计负债 (accrued liabilities) 企业所欠的尚未偿还的负债。

### 所有者权益账户

按照前面提及的扩展会计公式，所有者权益账户分为：所有者名下资本，所有者提取，收入和费用。

#### 13.11.4 分类帐

信息系统中所有账户的集合叫做分类帐。

#### 13.11.5 会计科目表

企业所使用的所有账户名称及其编号的列表叫做会计科目表。

具体编号中的科目号有国标的规定，然后子目号有的省市有规定，如果没有则自己内定。比如按照 [这个网页](#) 的介绍，资产类编号首位科目号是 1，负债是 2，所有者权益是 3 等等。

### 13.11.6 报告期间

比如有年度财务报表, 报告期为一年的会计报告, 或者还有一个月, 一季度等中期财务报告。

## 13.12 参考资料

1. [wiki 商业智能](#) [wiki ERP](#)
2. ERP 不花钱, 作者: 老肖 (OSCG) , 版本: 1.0 .
3. 开源智造编写的用户手册
4. Odoo Development Essentials , author: Daniel Reis , date: April 2015
5. [OpenERP v7 官方文档](#)
6. Odoo new API guideline Documentation , author: Nicolas Bessi , date: April 13, 2015 .
7. OpenERP 应用和开发基础, 作者: 老肖, 版本: 0.2 .
8. 会计学原理第 19 版作者: John J.Wild , Ken W. Shaw 等. 崔学刚译, 中国人民大学出版社.