

开源智造咨询有限公司(OSCG)

OPENERP 7.0 开发教程

V1.0

定稿日期: 2013 年 03 月 20 日
作者: 老肖

版本	日期	修改描述	撰写人	审查人	批准人
V1.0	2013/03/20	初版	老肖	King Wang	老肖
OS Consulting Group			参考: Document Reference		

1	前言	1
1.1	OPENERP 简介	1
1.2	内容概要	1
1.3	版权声明	1
2	OPENERP 架构	2
2.1	三层架构	2
2.2	模块结构	2
2.3	MVC 模式	3
3	OPENERP 开发演练	6
3.1	创建请假单对象	6
3.2	创建视图	8
3.3	创建菜单和 Action	12
3.4	测试	16
3.5	深入数据库	17
3.6	请假模块示例代码	18
4	OPENERP 对象	20
4.1	一切都是对象	20
4.2	访问 OpenERP 对象	20
4.3	再议 OPENERP 的对象	22
4.4	OPENERP 对象定义的属性	22
4.5	OPENERP 对象字段的定义	26
4.6	字段定义的参数	28
4.7	OpenERP 对象预定义方法	29
5	OPENERP 视图	32
5.1	视图定义	32
5.2	分组元素(GROUPING ELEMENTS)	32
5.3	数据元素(DATA ELEMENTS)	33
5.4	视图继承	35
5.5	视图事件	36
5.6	取得缺省值	37
6	菜单和动作 (MENU 和 ACTION)	38
6.1	菜单(MENUS)	38
6.2	动作(ACTION)	38
7	OPENERP 工作流	43
7.1	工作流定义	43
7.2	活动 (ACTIVITY) 定义	43
7.3	迁移 (TRANSITION) 的定义	44
8	OPENERP 报表开发	46

8.1	OpenERP 报表运行机制	46
8.2	RML 报表开发方法	47
8.2.1	概述	47
8.2.2	RML 语法格式	47
8.2.3	Report Action 配置	48
8.2.4	Parser	49
8.3	Aeroo 报表开发方法	49
8.3.1	概述	49
8.3.2	Aeroo 语法格式	49
8.3.3	AerooAction 配置	49
8.3.4	Parser	50
9	OPENERP 权限设置	51
10	OPENERP WEB 开发	53
10.1	Web 运行机制	53
10.2	Web 页面开发	53

1 前言

1.1 OpenERP 简介

OpenERP 是一款优秀的开源 ERP 软件。开源是说，软件完全公开，任何人可以自由下载软件，还可以自由下载软件的所有源代码，任意修改软件；软件本身没有任何秘密，没有任何收费。优秀是说，软件功能丰富，品质优秀。OpenERP 源自欧洲，技术先进，质量稳定。OpenERP 是欧洲中小型企业最受欢迎的 ERP 软件，拥有最多的用户数量。OpenERP 的开发历史超过 10 年，拥有 1700 多个功能模块，涵盖企业管理方方面面的业务流程，包括 Magento、TaoBao、Joomra 等网店软件集成接口。

和商业 ERP 软件相比，OpenERP 不需要任何版权费用，可以终身免费使用。OpenERP 也不按用户点数收费，系统上线后，无论增加多少终端用户，不需要增加任何软件费用。OpenERP 收取的是服务费，聘请 ERP 专业人员咨询、实施、培训以及安装维护 OpenERP 系统，提供服务的公司或人员需要收取必要的服务费用。

开源智造咨询有限公司（OSCG）是 OpenERP 香港和大陆地区的官方合作伙伴，我们提供专业的 OpenERP 系统咨询、实施、培训、二次开发、系统维护等服务。我们还提供基于 OpenERP 的在线 ERP 租用（SaaS）。我们的在线 ERP 按行业预配了很多模块和基础数据，只要导入自己的客户、产品等少量数据即可开始 ERP 系统。

开源智造咨询有限公司（OSCG）是大中华区最早的 OpenERP 合作伙伴之一，我们拥有最多的中国区 OpenERP 成功案例；我们的团队撰写了 OpenERP 中文教材，是大中华区的 OpenERP 权威专家；针对中国的财务制度、用户习惯、业务特点，我们对 OpenERP 做了大量中国特色的开发，拥有完整的适应中国区各种需求的功能模块。

1.2 内容概要

首先非常感谢您对 OpenERP 的关注。OSCG(开源智造)是中国 OpenERP 官方合作伙伴的先驱，公司总部在香港，在上海设有技术中心。OSCG 提供基于 OpenERP 的各种服务，包括实施、二次开发、咨询、培训等。您可以浏览我公司的网站，<http://www.oscg.cn>，或者 Email 联系我们：sales@oscg.com.hk。

本书是 OSCG 基于自身的开发实践，总结而成的 OpenERP 开发技术要点及经验之精华。内容包括 OpenERP 的架构技术、对象、视图、Action、Workflow、Wizard、报表、Webservice 等、Web 开发等 OpenERP 开发的全方面技术。

OpenERP 软件于 2013 年 1 月底发布了目前最新的版本 V7.0，在 V7.0 中，OpenERP 系统带来了许多创新，用户界面焕然一新，系统易用性极大提高。本书中，将我们总结的开发技术全面升级到了 V7.0 版本。

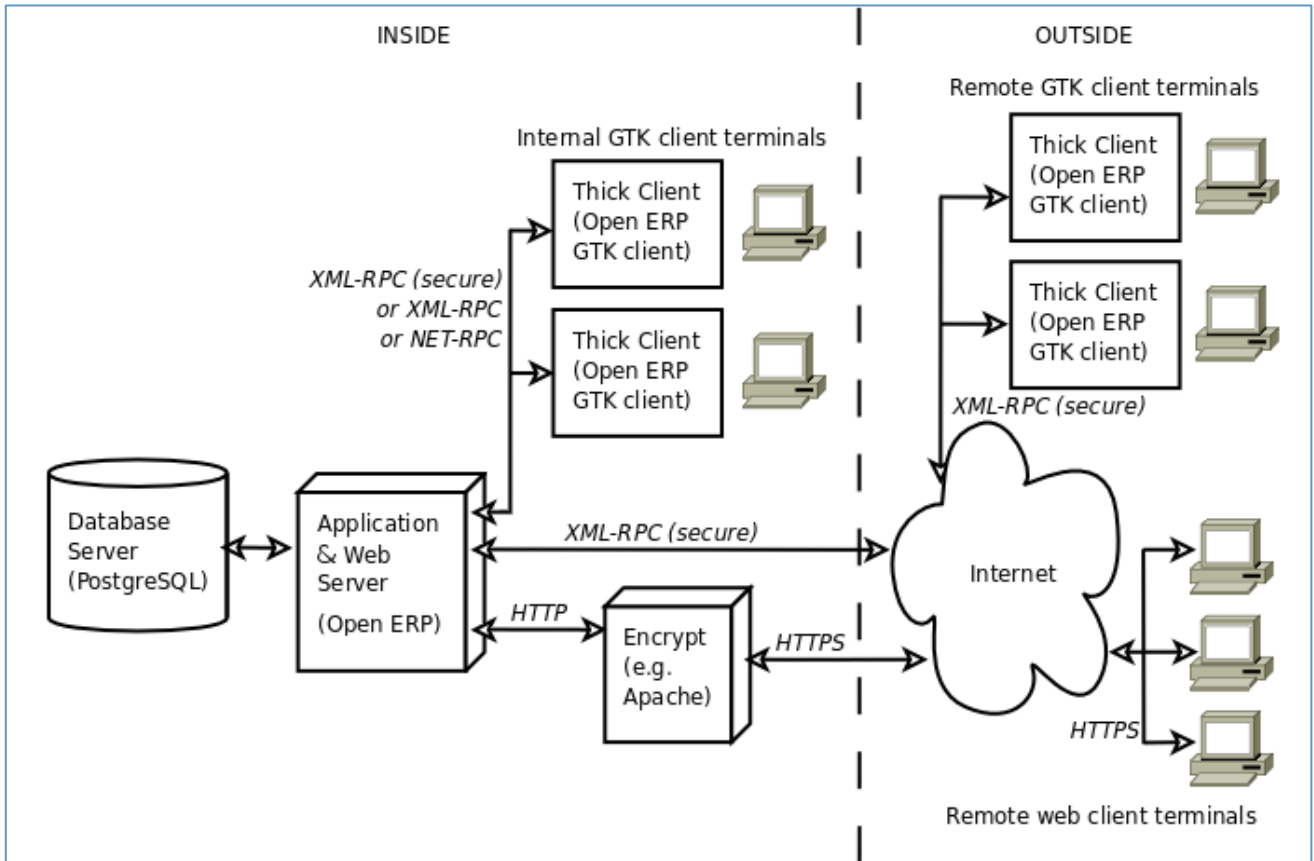
1.3 版权声明

本书版权属于香港开源智造咨询有限公司，任何人或机构可将本书内容用于学习、研究、教育，以及其他非商业性或非盈利性用途，但同时应遵守著作权法及其他相关法律法规的规定，不得侵犯本书作者的合法权利。除此以外，将本书任何内容用于其他用途时，须征得本书作者的书面许可，并支付报酬。

2 OPENERP 架构

2.1 三层架构

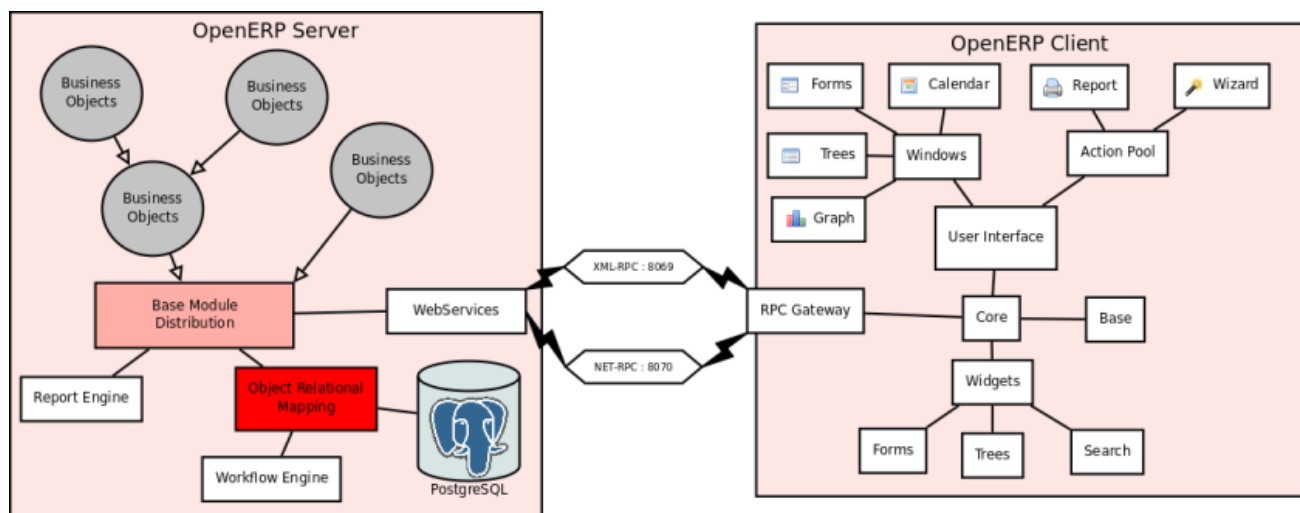
OpenERP 的安装部署，是典型的三层架构：数据库层、应用服务器层、客户层。



- ◆ **数据库：**OpenERP 采用开源数据库 PostgreSQL。PostgreSQL 研发历史超过 30 年，最初是由著名的 IT 学院伯克利大学研发的。是世界上最好和最著名的大型开源数据库。OpenERP 利用了 PostgreSQL 的相当多的特性，两者绑定相当紧密。因此，OpenERP 目前只支持 PostgreSQL，不支持其他数据库。
- ◆ **应用服务器：**OpenERP Server 承担应用服务器的角色。作为一个完整的企业应用服务器，OpenERP Server 包括数据访问层 ORM、界面展现层 View、Workflow 引擎、报表引擎，以及其他系统集成的 Webservice 接口。
- ◆ **客户端：**OpenERP 的客户端有两种，一种是 Web Client，就是浏览器。使用 Web Client，可以从任何有互联网的地方访问 OpenERP。另一种是 GTK Client，GTK Client 更多的部署在局域网内。

2.2 模块结构

OpenERP 的各种功能都是由一个个的模块提供，OpenERP V7.0 中，系统提供的标准模块有 170 多个，在 OpenERP Apps Store 上面还有近 2000 个模块。OpenERP 的模块，通常由这么一些对象组成：



- ◆ **Business Object:** 负责数据表的增删改查等操作。
- ◆ **View:** 将数据表的字段展现在窗口上。
- ◆ **Action:** 联系 Object 和 View 的 Controller。
- ◆ **Workflow:** 管理 Business Object 处理流程的工作流。
- ◆ **Report:** 将 Business Object 打印出来的报表。
- ◆ **Wizard:** 处理一些复杂的用户交互操作功能。

模块典型目录构成如下：



2.3 MVC 模式

当我们操作 OpenERP 的菜单时，通常是点击菜单（如销售 → 客户），跳出对象选择画面，当选择一条记录时，跳出对象编辑画面。



对象选择画面、对象编辑画面，在 OpenERP 里称为视图（View），选择画面是看板视图 (Kanban View) 和列表视图（Tree View），编辑画面是表单视图（Form View）。OpenERP 里的对象（Object），也叫 Model，相当于我们一般说的类（Class），对象总是对应到数据库里的数据表。例如业务伙伴对象，其对象名是“res.partner”，对应表名是“res_partner”。表里的一条记录，也就是对象的一个实例，叫资源（Resource）。



当点击菜单时，系统怎么知道应该跳出哪个画面，以及应该显示哪个对象的记录呢？把菜单和对应的对象、视图关联起来的是 Action。当用户点击菜单时，触发 Action，Action 调用对象的 Search 方法，从数据库取得记录（资源），Action 又创建视图，显示取得的数据。简单总结一下，OpenERP 的开发中，有如下一些重要概念：

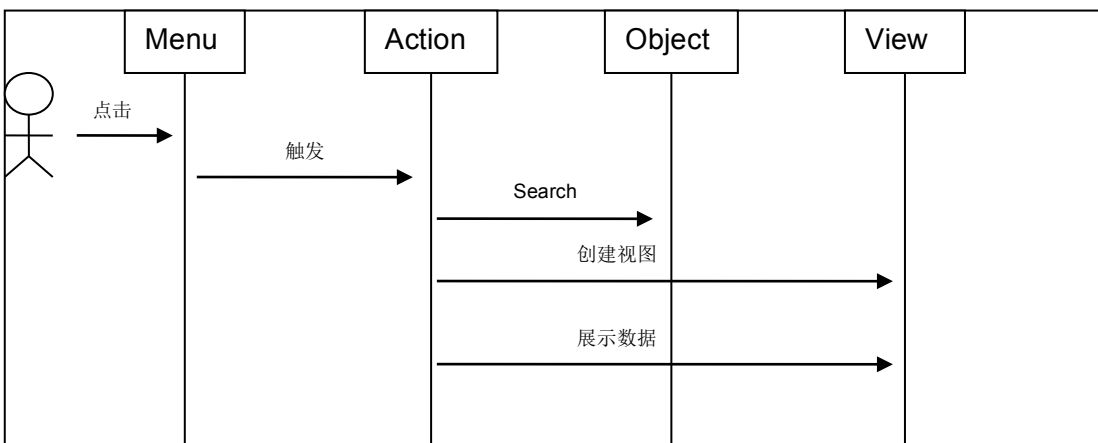
- ◆ **对象或模型（Object or Model）**：是一个 Python 的 Class，也对应到数据库的一张数据表，负责存取数据记录（Record），有 Search、Read、Write 等方法。OpenERP 在模块加载时，初始化模块中的所有对象，放入对象池。因此，数据库操作时，通常是先从对象池中取得对象，再调用对象的方法。下面分析一段典型的记录查找和读取代码。

```
def _get_admin_id(self, cr):
    if self.__admin_ids.get(cr.dbname) is None:
        ir_model_data_obj = self.pool.get('ir.model.data')
        mdid = ir_model_data_obj._get_id(cr, 1, 'base', 'user_root')
        self.__admin_ids[cr.dbname] = ir_model_data_obj.read(cr, 1, [mdid], ['res_id'])[0]['res_id']
    return self.__admin_ids[cr.dbname]
```

这段代码来自文件 `server\addons\base\res\ res_user.py`，它先从对象池取得对象 `'ir.model.data'`，该对象负责存取数据表 `ir_model_data`。而后调用该对象的方法 `_get_id` 取得 `admin` 用户的 `user_id`，再读入 `admin` 对应的 `user` 记录的 `id`。

- ◆ **视图（View）**：负责显示数据，最常见的视图是列表视图和表单视图。此外，还有看板、日历、甘特图、图形、流程图等几种视图，不同的视图以不同的方式展示数据。本章主要介绍列表和表单，另外几种视图以后介绍。
- ◆ **菜单（Menu）**：这个很直观，不用介绍了。
- ◆ **动作（Action）**：用户操作系统时（如点击菜单、点击画面右边的工具条上的按钮等），系统的响应动作。一个 **Action** 包含一个对象，包含若干个视图，通常每个 **Action** 都包含列表和表单两个视图。当 **Action** 被触发时，相应的视图被调出，展示相应的对象的数据。**Action** 有多种类型，最常见的是 **Act_Window**（窗口类型），窗口类型跳出一个窗口以显示数据。此外还有 **Report**（报表）、**Wizard**（向导）等类型。本章主要介绍窗口类型。

上述概念间的关联关系，参见如下操作序列图：



例如，当打开一个财务凭证时(对象 `account.invoice`)，客户端发生的动作链是：

- 1) 激发一个 **Action**，**Action** 要求打开 `account.invoice` 对象。**Action** 中包含了对象、视图、域条件（Domain，如只显示未支付的凭证）等数据。
- 2) 客户端询问服务端(通过 XML-RPC 协议)凭证对象定义了哪些视图，以及应显示什么数据。
- 3) 客户端呈现视图，展示数据。

3 OPENERP 开发演练

本节开发一个简单的请假申请功能，包括创建及编辑请假单，请假单查找。但暂不包括请假审批流程，审批流程将在后续章节开发。本节拟定的请假单包括如下信息：

- 申请人：申请人默认是当前登录用户，必填项。
- 请假天数：可以是小数，必填项。
- 开始日期：开始休假的日期，必填项。
- 请假事由：一段文本，描述请假事由，可以不填。

根据前一节的概念介绍，我们需要开发下述对象：

- 请假单对象：将请假单保存到数据库，以及从数据库查找请假单。
- 请假单视图：查找、编辑请假单的画面，包括列表和表单两个视图。
- 菜单：准备开发两级菜单，请假申请 → 请假单。点击“请假单”时，进入请假单列表视图，可以查找或创建请假单。
- 请假单动作(Action)：请假单动作把对象、视图、菜单关联到一起。

3.1 创建请假单对象

点击菜单：设置 → 技术 → 数据库结构 → 模型，点击新建按钮，创建请假单对象。

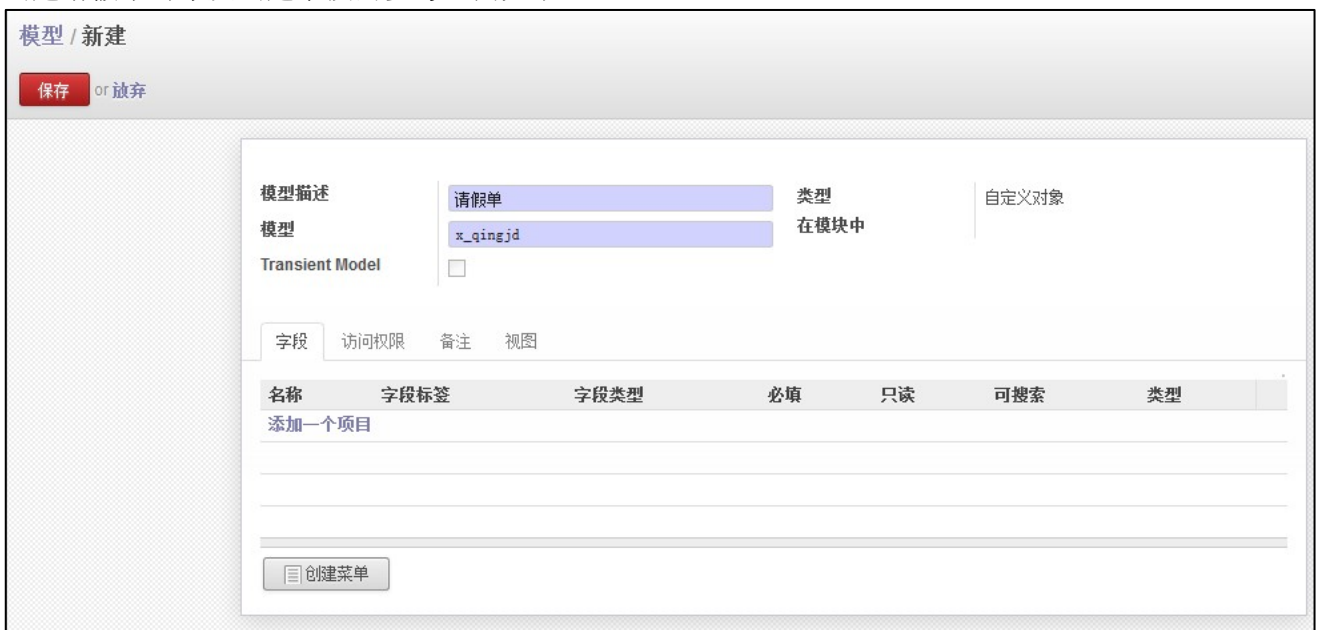
对象名：请假单

对象：x_qingjd

字段：包括四个字段，申请人（x_shenqr），请假天数（x_tians），开始日期（x_kaisrq），请假事由（x_shiyou）。

注意：从界面上创建对象时，对象及字段标识符必须是 x_ 开头，后面还会介绍编写代码来创建对象，编写代码创建对象时，就没有这个限制。

创建请假单对象及创建字段的参考画面如下：



点击“保存”，然后点击“编辑”

模型 / 请假单

保存 or 放弃

81 / 4

模型描述

请假单

类型

自定义对象

模型

x_qingjd

在模块中

Transient Model

☐

字段

访问权限

备注

视图

名称	字段标签	字段类型	必填	只读	可搜索	类型
添加一个项目						

创建菜单

点击“添加一个项目”逐个创建字段。

注意：一定要先创建对象，保存之后再编辑、创建字段。否则，一次性创建对象和字段，系统不能正确创建对象对应的数据库表，从而运行中会报错。这是 OE7.0 的一个小 bug。

模型 / 新建

保存 or 放弃

模型描述

请假单

类型

自定义对象

模型

x_qingjd

在模块中

Transient Model





☐

字段

访问权限

备注

视图

名称	字段标签	字段类型	必填	只读	可搜索	类型
x_shenqr	申请人	many2one	<input checked="" type="checkbox"/>	<input type="checkbox"/>	不可搜索	
x_tians	请假天数	float	<input checked="" type="checkbox"/>	<input type="checkbox"/>	不可搜索	
x_kaisrq	开始日期	date	<input checked="" type="checkbox"/>	<input type="checkbox"/>	不可搜索	
x_shiyou	请假事由	text	<input type="checkbox"/>	<input type="checkbox"/>	不可搜索	
添加一个项目						

创建菜单

创建：字段

名称

x_shenqr

类型

自定义字段

字段标签

申请人

属性

字段类型

many2one

必填

☒

对象关联

res.users

只读

☐

关联字段

可搜索

不可搜索

选择项目

可翻译

☐

大小

64

删除时

设为空 (NULL)

过滤条件

在模块中

序列化字段

组

保存并关闭

保存并新建

or 放弃

本画面中各个字段的含义，请参考后文中“OpenERP 对象”章节。

3.2 创建视图

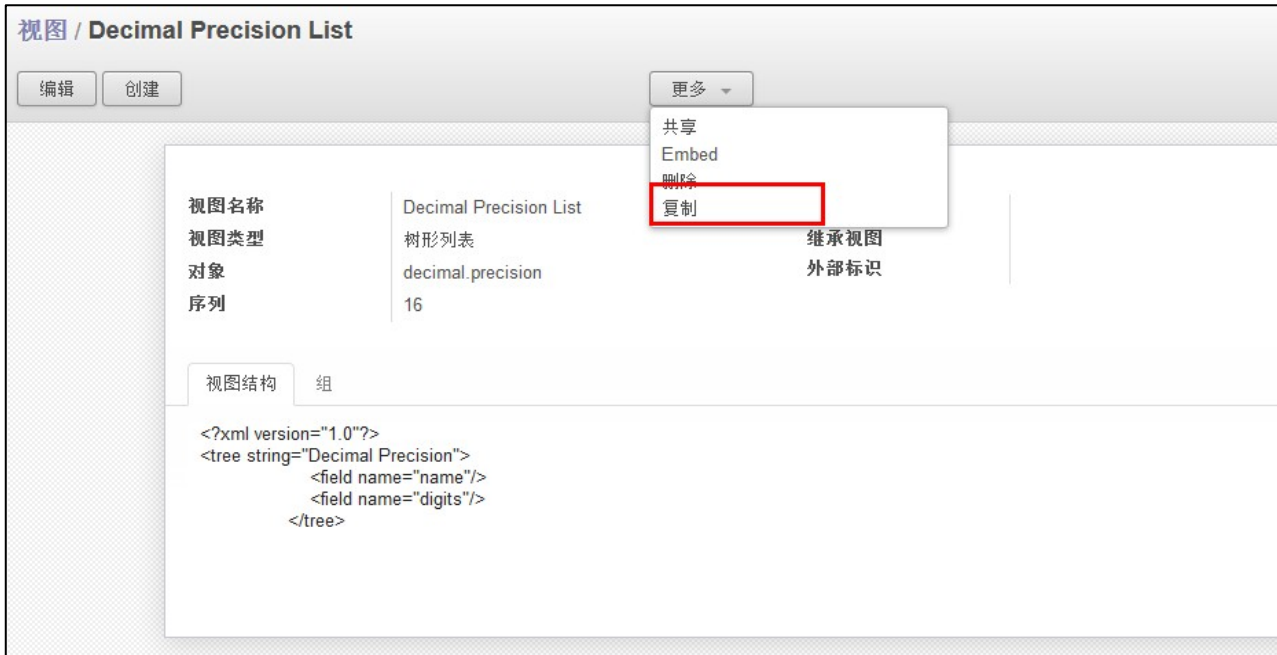
点击菜单：设置 → 技术 → 用户界面 → 视图，依次创建**请假单列表**和**请假单表单**两个视图。

注意：由于 OpenERP 7.0 系统的一点小 Bug，这里直接点击创建按钮创建视图，系统保存不了视图。操作上要先找一个同类型的视图，点击“复制”，得到一个新视图，再修改成想要的视图。

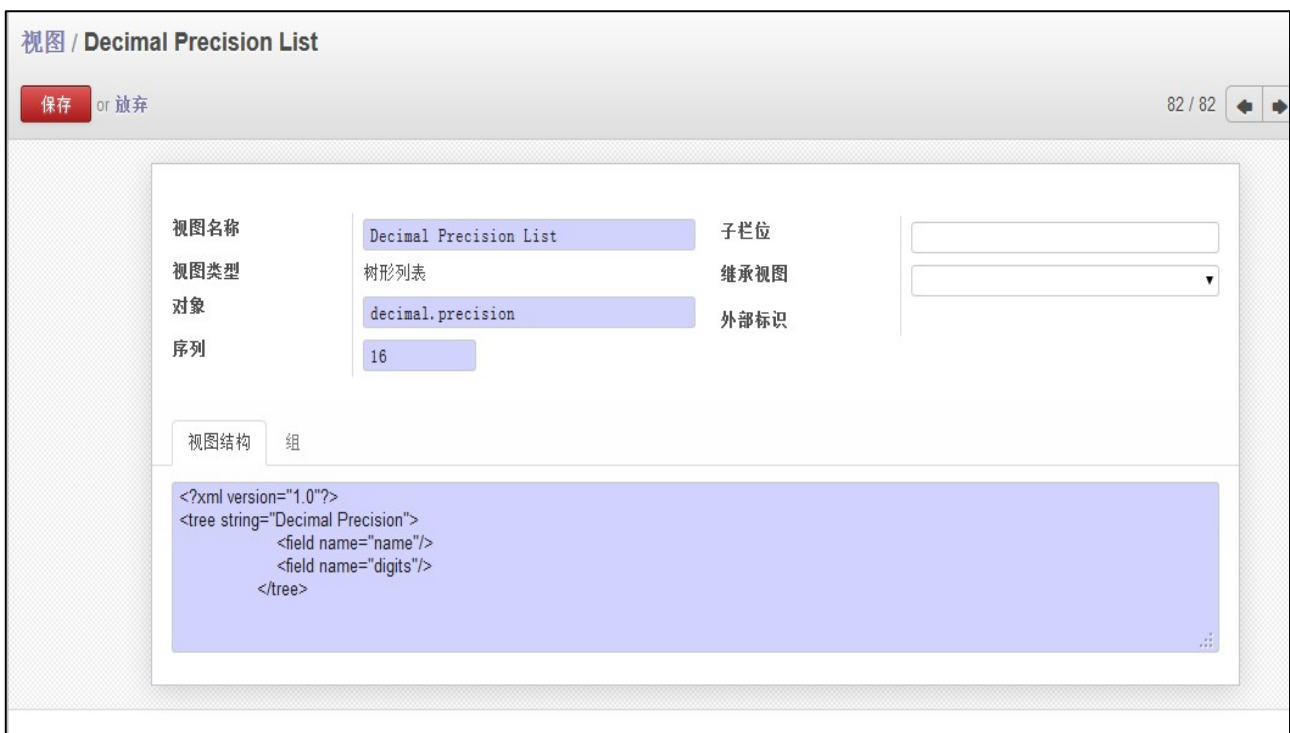
请假单列表

- 视图名称：请假单列表
- 视图类型：树形列表
- 对象：x_qingjd，即刚才创建的请假单对象，如果填写的对象标识符不正确的话，保存时会提示错误。子栏位和继承视图不填。
- 优先级：1，Action 通常有多个视图，优先级表示多个视图的显示顺序，数字越小越先显示。通常列表总是最先显示，其次是表单。
- 视图结构：视图结构是视图中最重要的一部分，也是最复杂的部分。视图结构定义要在视图中显示哪些字段，及显示属性（如只读、必填、隐藏等）。select="1"表示，在列表视图中，可以按该字段查找资源（记录）。

首先、从视图列表中选一个树形列表，复制之后修改成“请假单列表”视图。



点击“复制”，在原有基础上进行修改成我们需要的视图



本视图内容如下：

```
<?xml version="1.0"?>
<tree string="请假单">
  <field name="x_shenqr" select="1"/>
  <field name="x_tians" />
  <field name="x_kaisrq" select="1"/>
  <field name="x_shiyou" />
</tree>
```

视图 / 请假单列表

保存 or 放弃

视图名称	请假单列表	子栏位	
视图类型	树形列表	继承视图	
对象	x_qingjd	外部标识	
序列	16		

视图结构 组

```
<?xml version="1.0"?>
<tree string="请假单">
  <field name="x_shenqr" select="1"/>
  <field name="x_tians" />
  <field name="x_kaisrq" select="1"/>
  <field name="x_shiyou" />
</tree>
```

同理创建“请假单表单”视图。

视图名称：请假单表单

视图类型：表单

对象：x_qingjd

优先级：2

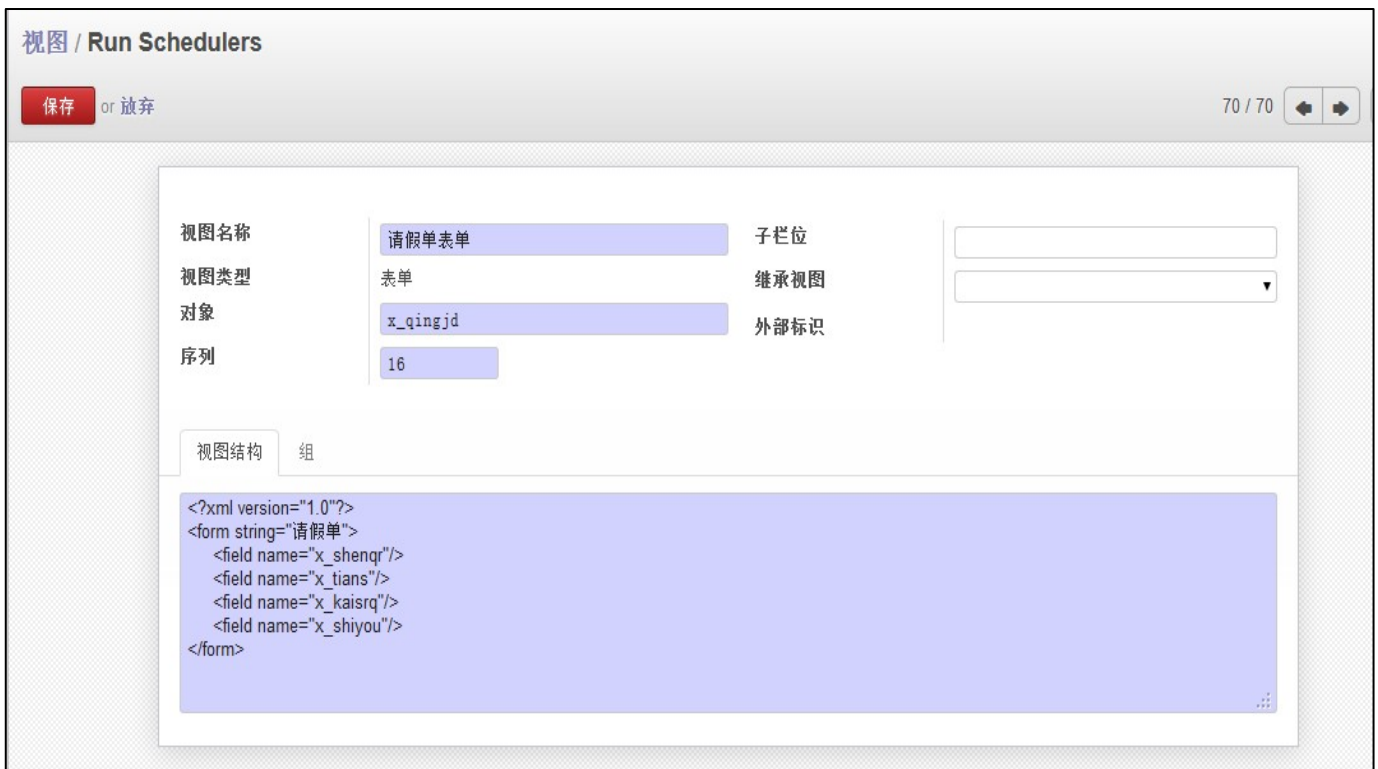
视图结构：本视图内容如下：

```
<?xml version="1.0"?>
<form string="请假单">
  <field name="x_shenqr"/>
  <field name="x_tians"/>
  <field name="x_kaisrq"/>
  <field name="x_shiyou"/>
</form>
```

首先、从试图列表中选一个表单



点击“复制”，在原有基础上进行修改成我们需要的表单



3.3 创建菜单和 Action

菜单 设置 → 技术 → 数据库结构 → 模型 进去，查找刚才创建的“请假单”对象，点击打开，在表单的下方有个“创建菜单”的按钮，点击进去。



模型 / 请假单

编辑 创建 打印 更多

模型描述 请假单 类型 自定义对象
模型 x_qingjd 在模块中
Transient Model ☐

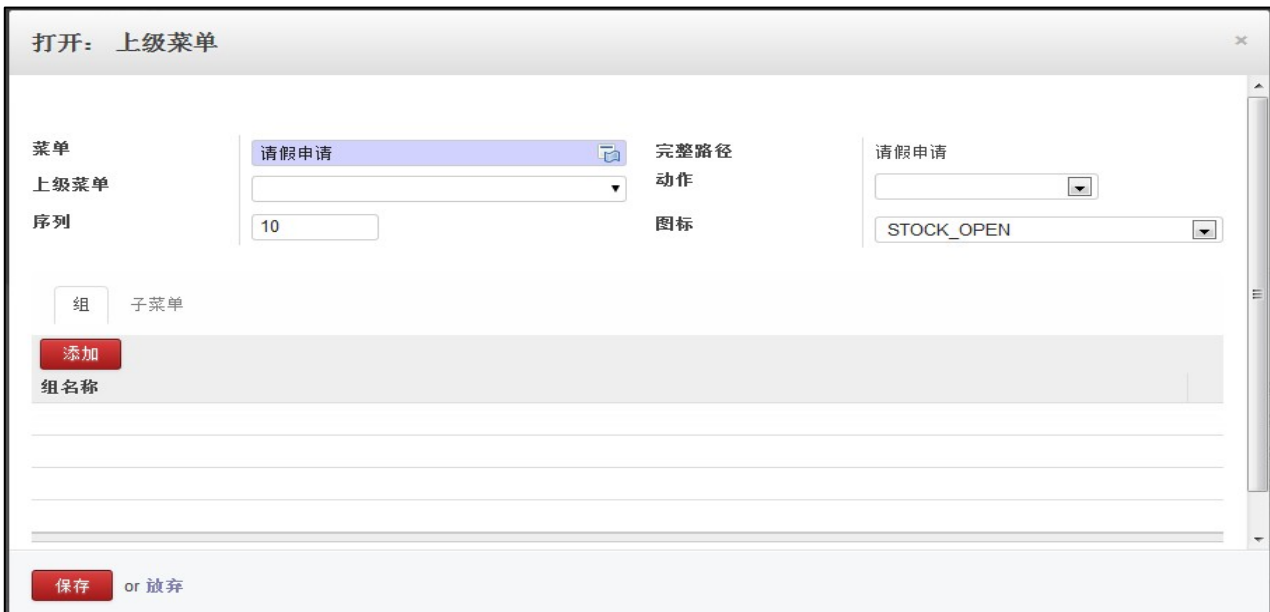
字段 访问权限 备注 视图

名称	字段标签	字段类型	必填	只读	可搜索	类型
x_kaisrq	开始日期	date	<input checked="" type="checkbox"/>	<input type="checkbox"/>	不可搜索	自定义字段
x_shenqr	申请人	many2one	<input checked="" type="checkbox"/>	<input type="checkbox"/>	不可搜索	自定义字段
x_shiyou	请假事由	text	<input type="checkbox"/>	<input type="checkbox"/>	不可搜索	自定义字段
x_tians	请假天数	float	<input checked="" type="checkbox"/>	<input type="checkbox"/>	不可搜索	自定义字段

创建菜单

进去后，点击父菜单右边的查找按钮，在父菜单查找画面，新建一个“请假申请”的父菜单。

- ◆ 完整路径：不用填，系统自动生成。
- ◆ 菜单：请假申请，用于菜单显示的文字
- ◆ 序号：在菜单列表中的显示位置，数字越小越显示在上位。
- ◆ 上级菜单：由于本菜单是顶级菜单，不要父菜单。
- ◆ 菜单功能：点击菜单时将触发的 Action，本菜单仅仅是父菜单，不触发 Action，故不填。
- ◆ 组：只有这里定义的组(group)才能看见本菜单，如果不填，表示任何组都可以看到。暂时不考虑菜单权限，故而不填。



打开：上级菜单

菜单 请假申请 完整路径 请假申请
上级菜单 动作
序列 10 图标 STOCK_OPEN

组 子菜单

添加

组名称

保存 or 放弃

点击“保存”

父菜单建好后，按下图创建“请假单”菜单。点击“创建菜单”按钮，系统会自动创建菜单和菜单对应的 Action。该 Action 访问请假单对象，依次调用请假单列表和请假单表单显示数据。

创建菜单

菜单名称

请假单

上级菜单

请假申请

创建菜单(M)

or 取消

再点击“创建菜单”，创建下级菜单

模型 / 请假单

编辑

创建

打印

更多

模型描述

请假单

类型

自定义对象

模型

x_qingjd

在模块中

Transient Model

☐

字段

访问权限

备注

视图

名称	字段标签	字段类型	必填	只读	可搜索	类型
x_kaisrq	开始日期	date	<input checked="" type="checkbox"/>	<input type="checkbox"/>	不可搜索	自定义字段
x_shenqr	申请人	many2one	<input checked="" type="checkbox"/>	<input type="checkbox"/>	不可搜索	自定义字段
x_shiyou	请假事由	text	<input type="checkbox"/>	<input type="checkbox"/>	不可搜索	自定义字段
x_tians	请假天数	float	<input checked="" type="checkbox"/>	<input type="checkbox"/>	不可搜索	自定义字段

创建菜单

创建菜单

菜单名称

请假单

上级菜单

请假申请/请假单

创建菜单(M)

or 取消

菜单项

创建

(1-3) 总 3

序列	菜单
10	请假申请
10	请假申请/请假单
10	请假申请/请假单/请假单1

菜单项 / 请假申请

编辑 创建 更多

菜单	上级菜单	完整路径	动作	图标
请假申请	10	请假申请	STOCK_OPEN	

组 子菜单

序列	菜单
10	请假单

菜单项 / 请假申请/请假单

编辑 创建 更多

菜单	上级菜单	完整路径	动作	图标
请假单	请假申请	请假申请/请假单	STOCK_OPEN	

组 子菜单

序列	菜单
10	请假单1

菜单项 / 请假申请/请假单/请假单1

编辑 创建 更多 ▾

菜单	请假单1	完整路径	请假申请/请假单/请假单1
上级菜单	请假申请/请假单	动作	请假单1
序列	10	图标	STOCK_INDENT

组 子菜单

序列	菜单

揭开 Action 的面纱

前面介绍过，是 Action 把对象、视图、菜单等各个元素集成到了一起，那么 Action 到底是个什么东西呢？

点击菜单：设置 → 技术 → 动作 → 窗口动作，在动作对象中输入“x_qingjd”查询，得到“请假单1”动作，点击打开。从 Action 编辑画面，可以看到，Action 关联了对象（x_qingjd）、视图（请假单列表和请假单表单），还有其他一些属性字段。

窗口动作 / 请假单1

保存 or 放弃

动作名称	请假单1	动作用途	
对象	x_qingjd	动作类型	ir.actions.act_window
源对象		目标窗口	当前窗口

常规设置 安全设置

视图	过滤
视图类型	表单
视图模式	tree,form
视图参照	
搜索视图引用	
	过滤条件值
	上下文值
	数量限制
	自动刷新
	自动搜索
	过滤

帮助

上述画面中各字段的含义，参见后文的“菜单和动作”章节。

3.4 测试

回到主菜单，画面上增加了菜单“请假申请 → 请假单 → 请假单 1”。点击请假单，进入列表视图，点击新建按钮，进入创建请假单画面。



点击“创建”



点击“保存”



3.5 深入数据库

在上述界面操作中，依次创建了对象、视图、菜单和系统动作，作成了“请假申请”的功能。在界面操作的背后，OpenERP 内部做了哪些动作呢？实际开发工作中，总是会碰到这样那样的问题，只有明白了其背后动作，才知道从哪里查找原因，解决问题。本节深入 OpenERP 数据库，探求背后内幕。

对象

当在界面上创建 x_qingjd 对象时，OpenERP 在数据库中新建了一张表 x_qingjd，用于保存请假单对象，如下：

x_qingjd Table

id	create_date	write_date	write_time	x_shenqr	x_tians	x_kaisrq	x_shiyou
[PK] integer	timestamp v	timestamp v	integer	integer	double p	date	text
1	2009-12-27 2	2009-12-28 0	1	3	1.5	2009-12-29	休年假

x_qingjd 对象及其字段的信息，也写入了数据表 ir_model，字段信息在 ir_model_fields。

ir_model_fields 通过 model_id 字段和 ir_model 外键关联。如下：

ir_model Table

id	model	name	state	info
[PK] serial	character va	character varying(6	character va	text
85	x_qingjd	请假单	manual	

ir_model_fields Table

id	model	model_id	name	relation	sequence	field_description	type
[PK] integer	character va	integer	character va	character va	integer	character va	character
587	x_qingjd	85	x_shenqr	res.users	0	申请人	many2one
588	x_qingjd	85	x_tians	NULL	0	请假天数	float
589	x_qingjd	85	x_kaisrq	NULL	0	开始日期	date
590	x_qingjd	85	x_shiyou	NULL	0	请假事由	text

视图

界面上创建的视图，写入在表 ir_ui_view，其中视图结构保存在 arch 字段中，是一段 XML 文本。如下：

ir_ui_view Table

id	name	model	type	arch	field_parameters	priority
[PK] serial	character va	character va	character	text	character va	integer
122	请假单列表	x_qingjd	tree	<?xml version="1.0"><tree><field name="x_shenqr" type="many2one" model="res.users" /></tree>		1
123	请假单表单	x_qingjd	form	<?xml version="1.0"><form><field name="x_shenqr" type="many2one" model="res.users" /></form>		2

菜单和动作

在界面上创建菜单（父菜单和子菜单）时，菜单信息保存在表 ir_uimenu，如下：

ir_ui_menu Table

id	parent_id	name	icon	create_date	write_date	sequence
[PK] integer	integer	character	character va	timestamp v	timestamp v	integer
103		请假申请	STOCK_OPEN	2009-12-27 2		10
104	103	请假单	STOCK_INDEX	2009-12-27 2		10

动作 (Action) 信息在表 `ir_act_window`, `ir_act_window_view`, `ir_values` 中。其中 Action 的基本信息在 `ir_act_window` 中, 字段 `res_model` 定义了和本 Action 关联的对象。Action 和视图的关联信息在 `ir_act_window_view` 中, 和菜单的关联信息在 `ir_values` 中。

`ir_act_window` Table

id	name	type	usage	view_id	res_model	view_type
[PK]	character varying(32)	character varying(32)	character	integer	character varying(32)	character
93	请假单	ir.actions.act_window			x_qingjd	form

`ir_values` Table

id	name	key	key2	model	value	meta	res_id
[PK]	character varying(32)	character varying(32)	character varying(32)	character varying(32)	text	text	integer
84	Menuitem	action	tree_but_open	ir.ui.menu	ir.actions.act_window,93		104

Action 的最大玄机在表 `ir_values` 中, 如本例, 表 `ir_values` 中的字段 `model` 和 `res_id` 表示, 本 Action 的触发菜单是 `ir_ui_menu` 表中的 `id=104` 的菜单项, 这正是“请假单”菜单。字段 `value` 表示本 Action 触发哪个动作, 本例中 `value='ir.action.act_window,93'`, 表示点击菜单时触发表 `ir_action_act_window` 中的 `id=93` 的 Action, 这个 Action 正是“请假单”动作。

至此, 我们基本上理清了 OpenERP 的背后动作:

- 1) 从表 `ir_ui_menu` 中读取菜单信息, 显示在画面上
- 2) 当用户点击菜单时, 从表 `ir_values` 中找到该菜单对应的 Action
- 3) 从表 `ir_act_window` 和 `ir_act_window_view` 中, 找到 Action 关联的对象和视图
- 4) 从表 `ir_model` 和 `ir_model_fields` 取得对象和字段信息, 构建访问对象(本例是 `x_qingjd`)的 SQL 文
- 5) 从表 `ir_ui_view` 中取得视图信息, 尤其是视图结构 XML, 根据视图结构 XML 构造画面
- 6) 将对象数据显示在视图画面上。

不管是在界面上开发, 还是编写代码开发, OpenERP 都是在上述表中写入对象、视图、Action、菜单信息。运行时从上述表中读取信息, 访问数据库, 构造画面, 显示数据, 处理用户动作。

3.6 请假模块示例代码

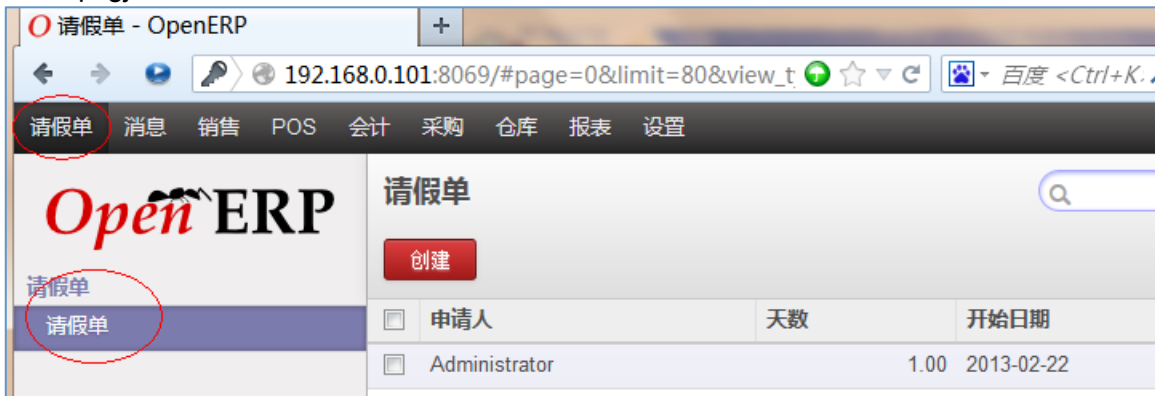
前一节从界面上创建了对对象、视图和菜单, 本节介绍直接写代码的方式创建请假功能模块, 实现同样的功能。从这里 (http://www.oscg.cn/?attachment_id=905) 下载示例代码, 解压该文件, 将 `qingjd` 目录拷贝到 OpenERP 安装目录下的 `openerp/addons` 下面。

点击菜单“设置→模块→更新”, 让 OpenERP 系统发现新模块“qingjd”;

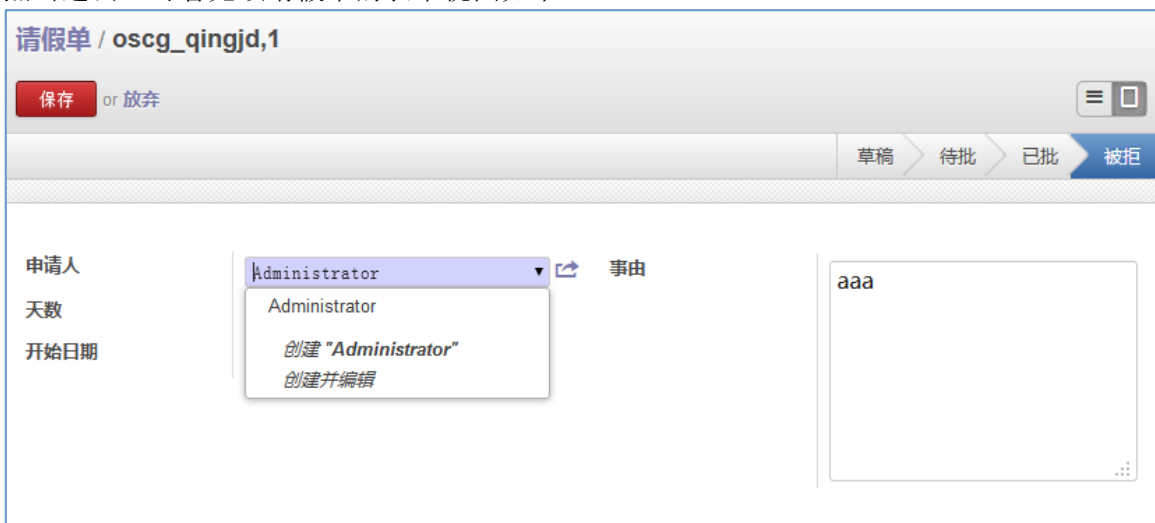
再点击菜单“设置→模块→可安装的模块”, 搜索 `qingjd`, 系统即可找到请假单模块, 如下图。



安装 qingjd 模块，该模块会在系统中增加一个“请假单”模块菜单，如下图所示。



点击进去，可看见该请假单的表单视图如下：



4 OPENERP 对象

4.1 一切都是对象

OpenERP 的所有资源(Resource)都是对象, 如 `menus`, `actions`, `reports`, `invoices`, `partners` 等等。换言之, 在 OpenERP 中, 一个菜单项, 一个弹出窗口, 其实都是一条数据库记录。OpenERP 运行时, 从数据库读出“菜单项”记录, 根据该记录的信息, 在屏幕上显示菜单项及其子菜单项。因此, 理论上, 可以不写代码, 而是直接修改 OpenERP 的数据库而编写功能菜单、查询窗口、动作按钮等实现业务功能开发。实际开发中, 通常是编写 XML 文件, 导入菜单、窗口、动作等编程元素, 实现功能开发。XML 文件比直接修改数据库或编写 SQL 语句更容易使用一些。

OpenERP 通过自身实现的对象关系映射(ORM, object relational mapping of a database)访问数据库。OpenERP 的对象名是层次结构的, 就是说可以使用"."访问树状对象, 如:

- `account.invoice`: 表示财务凭证对象。
- `account.invoice.line`: 表示财务凭证对象中的一个明细行对象。

通常, 对象名中, 第一级是模块名, 如: `account`, `stock`, `sale` 等。比之直接用 SQL 访问数据库, OpenERP 的对象的优势有, 1)直接使用对象的方法增、删、改数据库记录。因为 OpenERP 在基类对象中实现了常规的增、删、改方法, 因而, 普通对象中不需要写任何方法和代码就具备增、删、改数据库记录的功能。2)对于复杂对象, 只需操作一个对象即可访问多张数据表。如 `partner` 对象, 它的信息实际上存储在多张数据表中(`partner address`, `categories`, `events` 等等), 但只要通过"."操作即可访问所有关联表(如, `partner.address.city`), 简化了数据库访问。

注意, 在其他编程语言或开发平台(如 Java or JavaEE)中, 一个对象(Object)通常和数据库中一条记录(Record)相对应。但是, OpenERP 的对象其实是一个 Class, 它和一个数据表(Table)对应, 而不是和一个记录(Record)对应。在 OpenERP 中, 数据库记录(Record)通常叫资源(Resource)。因为 Object 操作的是数据表, OpenERP 的对象的方法(Method)中, 几乎每个方法都带有参数 `ids`, 该参数是资源(Resource or Record)的 ID (在 OpenERP 中 ID 是主键)列表, 通过该 `ids` 就可以操作具体的 Record 了。

4.2 访问 OpenERP 对象

OpenERP 提供了三种方式执行对象的方法(Method), 每种方式都是先取得对象, 然后调用对象的方法。三种方式是, 1)直接使用对象, 2)通过 `netservice` 使用对象, 3)通过 `xmlrpc` 使用对象。

直接使用对象: 这种方式最简单, 这种方式只能在 OpenERP Server 端使用, 编写 OpenERP 的模块时候, 通常使用这种方式。这个方式的内部实现原理是, OpenERP 加载模块(不是安装, 是启动时加载已安装模块)时, 会将创建模块中的对象实例, 对象实例以对象名为关键字, 存储在对象池(pool)中。此方式是, 从 pool 中取得对象, 而后调用对象的方法。

这个方式的一般调用形式是:

```
obj=self.pool.get('name_of_the_object')
obj.name_of_the_method(parameters_for_that_method)
```

第一行代码从对象池中取得对象实例, 第二行代码调用对象的方法。

Netservice 方式: 这个方式和直接使用对象的方式是类似的, 只是不以对象的方式呈现, 而是以“服务”(Service)的方式呈现。这种方式也只能在 OpenERP Server 端使用, 即调用程序和 OpenERP Server 程序在同一个 Python 虚拟机上运行。这个方式的内部实现原理是, 类似对象池, OpenERP 有一个全局变量的服务池: `SERVICES`, 该变量位于 `bin\netsvc.py`。有一些对象, 它在创建时(`__init__`方法中)将自己提供的服务登记在服务池中, 并暴露自己的服务方法(即该服务可供调用的

method)。和对象池不同的是，服务可以有选择性的暴露自己的方法。OpenERP 的工作流(Workflow)、报表 (Report)都以服务的形式暴露自己的方法，关于 OpenERP 可供使用的服务有哪些，将在以后介绍。这个方式调用形式如下：

```
service = netsvc.LocalService("object_proxy")
result = service.execute(user_id, object_name, method_name, parameters)
```

第一行指定服务名取得服务，"object_proxy"是 osv.osv 对象初始化时注册的一个服务，这个服务可用于调用 OpenERP 的几乎所有对象 (准确的说是所有从 osv.osv 派生的对象)。“object_proxy”服务用于调用对象的方法。第二行是其调用格式，execute 是该服务暴露的一个服务方法，该方法的参数说明如下：

user_id: 用户 id，以用户名、密码登录后取得的 id。

object_name: 对象名，欲访问的对象的名称，如"res.partner"等。

method_name: 方法名，欲调用的方法的名称，如"create"等。

parameters: 方法的参数。

XML-RPC 方式：这个方式相当灵活，它以 HTTP 协议远程访问对象，因此，能在本机、局域网、广域网范围调用 OpenERP 的对象的方法。该方式的调用形式是：

```
sock = xmlrpclib.ServerProxy('http://server_address:port_number/xmlrpc/object')
result = sock.execute(user_id, password, object_name, method_name, parameters)
```

参数说明如下：

server_address: 运行 OpenERP Server 的机器的 IP 或域名。

port_number: OpenERP Server 的 xmlrpc 调用端口，缺省情况是 8069。

execute 的参数和 Netservice 方式相同，只是多了个 password 参数，该参数即用户的登录密码。

XML-RPC 方式参考例子。这个例子以 xmlrpc 方式调用 OpenERP 的对象 res.partner，在数据库中插入一条业务伙伴及其联系地址记录。因为含有中文，测试时注意代码文件保存成 utf-8 格式：

```
# -*- encoding: utf-8 -*-
import xmlrpclib #导入 xmlrpc 库，这个库是 python 的标准库。
username = 'admin' #用户登录名
pwd = '123' #用户的登录密码，测试时请换成自己的密码
dbname = 'casel' #数据库帐套名，测试时请换成自己的帐套名
# 第一步，取得 uid
sock_common = xmlrpclib.ServerProxy('http://localhost:8069/xmlrpc/common')
uid = sock_common.login(dbname, username, pwd)
#replace localhost with the address of the server
sock = xmlrpclib.ServerProxy('http://localhost:8069/xmlrpc/object')

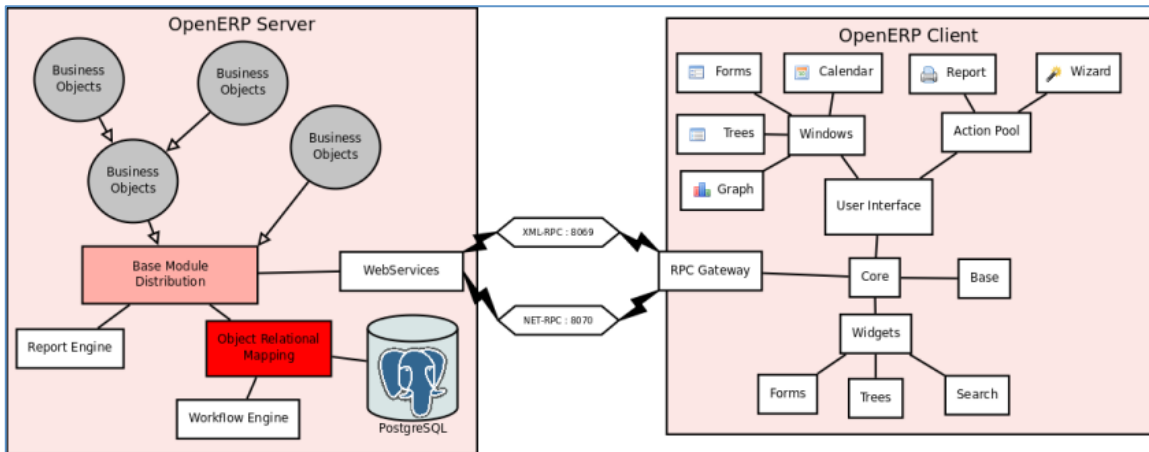
# 调用 res.partner 对象的 create 方法在数据库中插入一个业务伙伴
partner = {
    'name': 'shine-it',
    'lang': 'zh_CN',
}
partner_id = sock.execute(dbname, uid, pwd, 'res.partner', 'create', partner)

# 下面再创建业务伙伴的联系地址记录
address = {
    'partner_id': partner_id,
    'type': 'default',
    'street': '浦东大道 400 号',
    'zip': '200000',
    'city': '上海市',
}
```

```
'phone': '021-88888888',
}
address_id = sock.execute(dbname, uid, pwd, 'res.partner.address', 'create', address)
```

4.3 再议 OPENERP 的对象

在《OpenERP 应用和开发基础》的第一章中提过 OpenERP 的架构如下图所示：



这个架构图中，Server 端的 Business Object 就是这里重点解说的内容。通过上面的解说，可以这样通俗的理解 OpenERP 的对象：

每个对象就是一个代码块，包含了数据表操作(增删改查)的代码。OpenERP Server 好比是一个代码池，里面装满了代码块。通过对象池、服务池、xmlrpc 等方式，可以取得代码块位置(或者专业一点，叫指针)，然后调用代码块的方法，操作数据库。对象的代码什么时候装入“代码池”呢？每个对象定义的后面都有一行：`name_of_the_object()`，这一行实际上是创建对象实例，实例创建好以后就装到了代码池，这个装入的过程在对象的基类(`osv.osv`)中完成。对象的基类(`osv.osv`)已实现了增删改查等常规数据表操作方法，因此，只要定义好对象的字段，即使不写任何代码，该对象已经具备增删改查数据表的能力。

4.4 OPENERP 对象定义的属性

OpenERP 的对象定义的一般形式如下。

```
class name_of_the_object(osv.osv):
    _name = 'xxx'
    .....
name_of_the_object()

#Sample:
class qingjd(osv.osv):
    _name = 'qingjia.qingjd'
    _description = '请假单'
    _columns = {
        'shenqr': fields.many2one('hr.employee', '申请人', required=True),
    }
qingjd()
```

对象定义的完整属性如下：

必须属性：

`_name`
`_columns`

可选属性:

`_table`
`_description`
`_defaults`
`_order`
`_rec_name`
`_auto`
`_constraints`
`_sql_constraints`
`_inherit`
`_inherits`

下面详细解说各个属性。

`_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, **OpenERP** 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, **OpenERP** 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**。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`: 定义对象的字段, 系统会为这里定义的每个字段在数据库表中创建相应的字段。关于字段 (**Fields**) 的定义, 参见后文。

`_constraints`: 定义于对象上的约束 (**constraints**), 通常是定义一个检查函数, 关于约束的详细说明, 参见后文。

`_defaults`: 定义字段的缺省值。当创建一条新记录 (**record or resource**) 时, 记录中各字段的缺省值在此定义。

`_description`: 对象说明性文字, 任意文字。

`_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。

`_name`: 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**"。

`_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。详细说明参见后文。

`_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`**。

`_sequence`: 数据库表的 **id** 字段的序列采集器, 缺省值为: **None**。**OpenERP** 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。

`_sql`: **`_auto`** 为 **True** 时, 可以在这里定义创建数据库表的 **SQL** 语句。不过 **5.0** 以后好像不支持了, 不建议使用。

`_sql_constraints`: 定义于对象上的约束 (**constraints**), 和 **SQL** 文中的约束类似, 关于约束的详细说明, 参见后文。

`_table`: 待创建的数据库表名, 缺省值是和 **`_name`** 一样, 只是将“**.**”替换成“**_**”。

`_inherits`:

`_inherit`: **`_inherits`** 和 **`_inherit`** 都用于对象的继承, 详细说明参见后文。

`_constraints`

`_constraints` 可以灵活定义 OpenERP 对象的约束条件，当创建或更新记录时，会触发该条件，如果条件不符合，则弹出错误信息，拒绝修改。

`_constraints` 的定义格式：

`[(method, 'error message', list_of_field_names), ...]`

· `method`: 是对象的方法，该方法的格式为：`def _name_of_the_method(self, cr, uid, ids): -> True|False`

· `error message`: 不符合检查条件（`method` 返回 `False`）时的错误信息。

· `list_of_field_names`: 字段名列表，这些字段的值会出现在 `error message` 中。通常列出能帮助用户理解错误的字段。

`_constraints` 的例子：

```
def _constraint_sum(self, cr, uid, ids):
    cr.execute(' SELECT a.currency_id
                FROM account_move m, account_move_line l, account_account a
                WHERE m.id=l.move_id AND l.account_id=a.id AND m.id IN ('+', '.join(map(str, ids))+')
                GROUP BY a.currency_id')
    if len(cr.fetchall()) >= 2:
        return True
    cr.execute(' SELECT abs(SUM(l.amount))
                FROM account_move m LEFT JOIN account_move_line l ON (m.id=l.move_id)
                WHERE m.id IN ('+', '.join(map(str, ids))+')')
    res = cr.fetchone()[0]
    return res < 0.01
```

```
_constraints = [
    (_constraint_sum, 'Error: the sum of all amounts should be zero.', ['name'])
]
```

`_sql_constraints` 和 `_order`

`_sql_constraints` 定义数据表的约束条件，其格式如下例所示。

```
_sql_constraints = [
    ('code_company_uniq', 'unique (code,company_id)', 'The code of the account must be unique per company !')
]
```

本例的 `_sql_constraints` 会在数据表中增加下述约束：

`CONSTRAINT ObjectName_code_company_uniq UNIQUE(code, company_id)`

`_order` 在对象的 `search` 或 `read` 方法中的 `select` 语句上加上 "Order" 子句，如 `_order = 'name desc, account_id'`，对应 SQL 文的 Order 子句：`order by name desc, account_id`。

`_defaults`

`_defaults` 属性用于定义字段的缺省值，其格式为：

```
_defaults:
{
    'name_of_the_field':function, ...
}
```

`function` 的返回值作为 'name_of_the_field' 字段的缺省值。`function` 格式是：`function(obj, cr, uid, context)`，返回值必须是简单类型，如 `boolean`, `integer`, `string` 等。下面是 `_defaults` 的例子。

```
_defaults = {
```



```
'date_order': lambda *a: time.strftime('%Y-%m-%d'),
'state': lambda *a: 'draft',
'user_id': lambda obj, cr, uid, context: uid
}
```

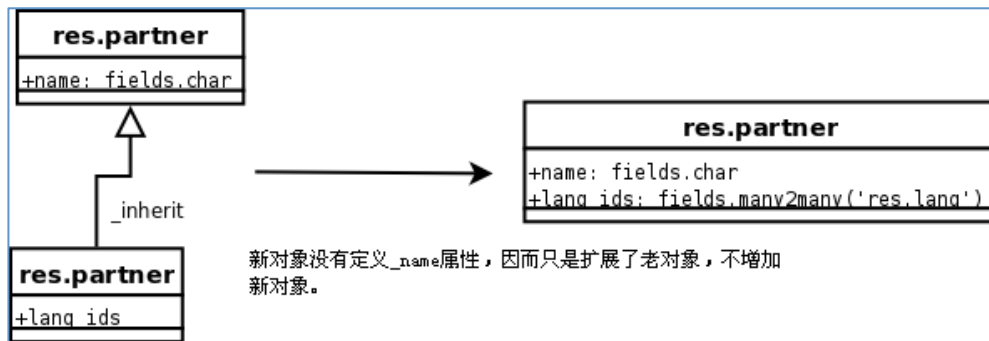
lambda 是 Python 的行函数, "lambda obj, cr, uid, context: uid" 等同于下述函数:

```
def func(obj, cr, uid, context):
    return uid
```

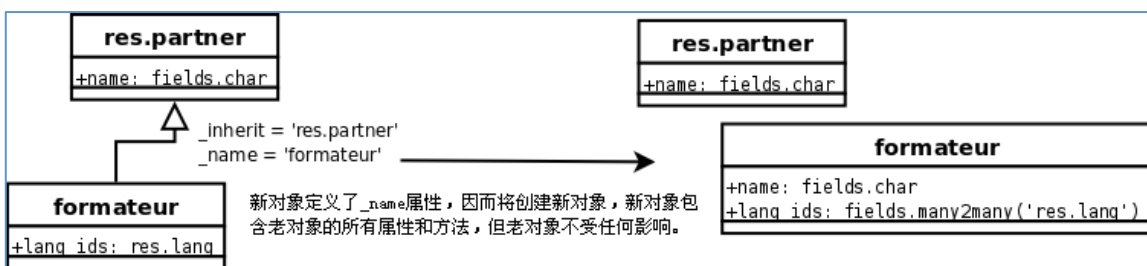
在 V6 中字典的值可以不是函数, 就比如在 V5 中我们必须这样来定义: `_defaults= {'state': lambda *a: 'draft'}` 而在 V6 中可以这样来: `_defaults = {'state': 'draft'}`

_inherit 和 _inherits

`_inherit` 继承有两种情况, 1) 如果子类中不定义 `_name` 属性, 则相当于在父类中增加一些字段和方法, 并不创建新对象。2) 如果子类中定义 `_name` 属性, 则创建一个新对象, 新对象拥有老对象的所有字段和方法, 老对象不受任何影响。两种情况的示例及继承关系的图示见下面。

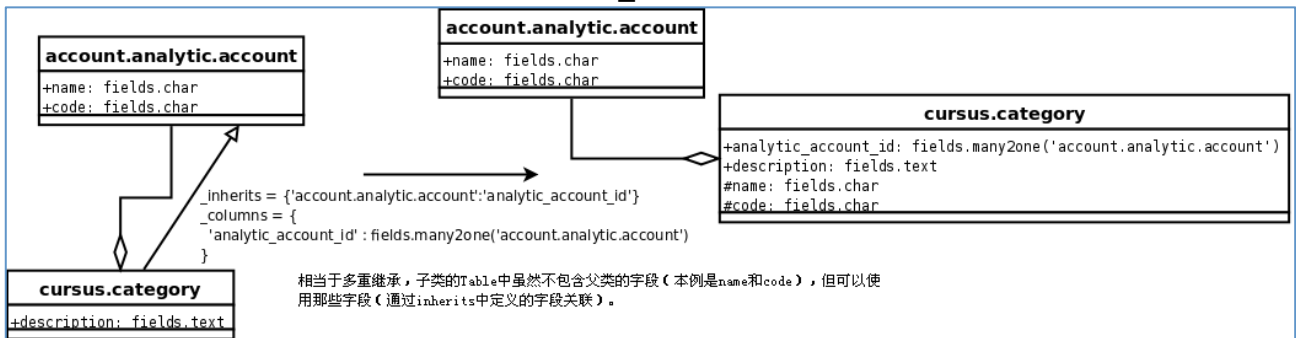


```
class res_partner_add_langs(osv.osv):
    _inherit = 'res.partner'
    _columns = {
        'lang_ids': fields.many2many('res.lang', 'res_lang_partner_rel', 'partner_id', 'lang_id',
        'Languages'),
    }
    res_partner_add_langs()
```



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

`_inherits` 相当于多重继承。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。`_inherits` 的示例及图示见下图。



```

class cursus_category(osv.osv):
    _name = 'cursus.category'
    _inherits = {'account.analytic.caccount': 'analytic_caccount_id'}
    _columns = {
        'analytic_caccount_id': fields.many2one('account.analytic.caccount', 'ID'),
    }
    cursus_category()
  
```

4.5 OPENERP 对象字段的定义

OpenERP 对象支持的字段类型有，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。下面逐一说明。

boolean: 布尔型(true, false)

integer: 整数。

float: 浮点型，如 'rate': fields.float('Relative Change rate', 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

function: 函数型，该类型的字段，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。

· 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。

· fnct 是函数或方法，用于计算字段值。如果 method = true, 表示 fnct 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。

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

· fnct_search 定义该字段的搜索行为。如果 method = true, 其格式为：def fnct_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 `fcnt`。

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, OpenERP 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` 或 `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` 或 `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>
```

4.6 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。

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”。

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:

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:

`'default_credit_account_id': fields.many2one('account.account', 'Default Credit Account',
domain="['<code>type</code>','!=','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。

4.7 OpenERP 对象预定义方法

每个 OpenERP 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:

基本方法: `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)
```

缺省值存取方法: `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)
```

特殊字段操作方法: `perm_read`, `perm_write`

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

字段(**fields**)和视图(**views**)操作方法: `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)
```

记录名字存取方法: `name_get`, `name_search`

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

缺省值存取方法: `default_get`, `default_set`

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

create 方法：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): OpenERP 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'})`

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《OpenERP 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。

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 (optional):** 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, ...}, ...]

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 **partner** 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。

write 方法: 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'})`

unlink 方法: 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。

default_get 方法: 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'])`

default_set 方法: 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**

5 OPENERP 视图

5.1 视图定义

示例

```
<?xml version="1.0"?>
<terp>
  <data>
    [view definitions]
  </data>
</terp>
```

The view definitions contain mainly three types of tags:

- <record> tags with the attribute model="ir.ui.view", which contain the view definitions themselves
- <record> tags with the attribute model="ir.actions.act_window", which link actions to these views
- <menuitem> tags, which create entries in the menu, and link them with actions

New : You can precise groups for whom the menu is accessible using the groups attribute in menuitem tag.

New : You can now add shortcut using the shortcut tag.

Example

```
<shortcut name="Draft Purchase Order (Proposals)" model="purchase.order" logins="demo" menu="m"/>
```

Note that you should add an id attribute on the menuitem which is referred by menu attribute.

```
<record model="ir.ui.view" id="v">
  <field name="name">sale.order.form</field>
  <field name="model">sale.order</field>
  <field name="priority" eval="2"/>
  <field name="arch" type="xml">
    <form string="Sale Order">
```

```
.....
  </form>
</field>
</record>
```

Default value for the priority field : 16. When not specified the system will use the view with the lower priority.

5.2 分组元素(GROUPING ELEMENTS)

Separator

视图上增加一条分割线，例：

```
<separator string="Links" colspan="4"/>
```

The string attribute defines its label and the colspan attribute defines his horizontal size (in number of columns).

Notebook

<notebook>: 定义Tab页。With notebooks you can distribute the view fields on different tabs (each one defined by a page tag). You can use the tabpos properties to set tab at: up, down, left, right.

示例：

```
<notebook colspan="4">....</notebook>
```

Group

<group>: groups several columns and split the group in as many columns as desired.

- colspan: the number of columns to use
- rowspan: the number of rows to use
- expand: if we should expand the group or not
- col: the number of columns to provide (to its children)
- string: (optional) If set, a frame will be drawn around the group of fields, with a label containing the string. Otherwise, the frame will be invisible.

Example

```
<group col="3" colspan="2">
  <field name="invoiced" select="2"/>
```



```
<field name="customer" select="1"/>
<field domain="[('domain', '=', 'partner')]" name="title"/>
<field name="lang" select="2"/>
<field name="supplier" select="2"/>
</group>
<notebook colspan="4">
<page string="General">
  <field colspan="4" mode="form,tree" name="address" nolabel="1" select="1">
    <form string="Partner Contacts">
      <field name="name" select="2"/>
      <field domain="[('domain', '=', 'contact')]" name="title"/>
      <field name="function"/>
      <field name="type" select="2"/>
      <field name="street" select="2"/>
      <field name="street2"/>
      <newline/>
      <field name="zip" select="2"/>
      <field name="city" select="2"/>
      <newline/>
      <field completion="1" name="country_id" select="2"/>
      <field name="state_id" select="2"/>
      <newline/>
      <field name="phone"/>
      <field name="fax"/>
      <newline/>
      <field name="mobile"/>
      <field name="email" select="2" widget="email"/>
    </form>
    <tree string="Partner Contacts">
      <field name="name"/>
      <field name="zip"/>
      <field name="city"/>
      <field name="country_id"/>
      <field name="phone"/>
      <field name="email"/>
    </tree>
  </field>
  <separator colspan="4" string="Categories"/>
  <field colspan="4" name="category_id" nolabel="1" select="2"/>
</page>
<page string="Sales & Purchases">
  <separator string="General Information" colspan="4"/>
  <field name="user_id" select="2"/>
  <field name="active" select="2"/>
  <field name="website" widget="url"/>
  <field name="date" select="2"/>
  <field name="parent_id"/>
  <newline/>
</page>
<page string="History">
  <field colspan="4" name="events" nolabel="1" widget="one2many_list"/>
</page>
```

Button

`<button/>`: add a button using the string attribute as label. When clicked, it can trigger methods on the object, workflow transitions or actions (reports, wizards, ...).

- string: define the button's label
- confirm: the message for the confirmation window, if needed. Eg: confirm="Are you sure?"
- name: the name of the function to call when the button is pressed. In the case it's an object function, it must take 4 arguments: cr is a database cursor
 - uid is the userID of the user who clicked the button
 - ids is the record ID list
 - **args is a tuple of additional arguments
- states: a comma-separated list of states (from the state field or from the workflow) in which the button must

appear. If the states attribute is not given, the button is always visible.

- type: this attribute can have 3 values – “workflow” (value by default): the function to call is a function of workflow
 - “object”: the function to call is a method of the object
 - “action”: call an action instead of a function

Example

```
<button name="order_confirm" states="draft" string="Confirm Order" icon="gtk-execute"/>
```

Label

Adds a simple label using the string attribute as caption.

Example

```
<label string="Test"/>
```

New Line

Force a return to the line even if all the columns of the view are not filled in.

Example

```
<newline/>
```

5.4 视图继承

When you create and inherit objects in some custom or specific modules, it is better to inherit (than to replace) from an existing view to add/modify/delete some fields and preserve the others.

Example

```
<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>
```

The inheritance engine will parse the existing view and search for the the root nodes of

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

It will append or edit the content of this tag. If this tag has some attributes, it will look for the matching node, including the same attributes (unless position).

This will add a page to the notebook of the res.partner.form view in the base module.

You can use these values in the position attribute:

- inside (default): your values will be appended inside this tag
- after: add the content after this tag
- before: add the content before this tag
- replace: replace the content of the tag.

Second Example

```
<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">
    <page string="Extra Info" position="replace">
      <field name="relation_ids" colspan="4" nolabel="1"/>
    </page>
  </field>
</record>
```

Will replace the content of the Extra Info tab of the notebook by one ‘relation_ids’ field.

The parent and the inherited views are correctly updated with –update=all argument like any other views.

To delete a field from a form, an empty element with position=”replace” attribute is used. Example:

```
<record model="ir.ui.view" id="view_partner_form3">
  <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">
    <field name="lang" position="replace"/>
  </field>
</record>
```

Take into account that only one position="replace" attribute can be used per inherited view so multiple inherited views must be created to make multiple replacements.

5.5 视图事件

On Change

The on_change attribute defines a method that is called when the content of a view field has changed.

This method takes at least arguments: cr, uid, ids, which are the three classical arguments and also the context dictionary. You can add parameters to the method. They must correspond to other fields defined in the view, and must also be defined in the XML with fields defined this way:

```
<field name="name_of_field" on_change="name_of_method(other_field'_1_', ..., other_field'_n_')"/>
```

The example below is from the sale order view.

You can use the 'context' keyword to access data in the context that can be used as params of the function.:

```
<field name="shop_id" select="1" on_change="onchange_shop_id(shop_id)"/>
```

```
def onchange_shop_id(self, cr, uid, ids, shop_id):
    v={}
    if shop_id:
        shop=self.pool.get('sale.shop').browse(cr,uid,shop_id)
        v['project_id']=shop.project_id.id
    if shop.pricelist_id.id:
        v['pricelist_id']=shop.pricelist_id.id
    v['payment_default_id']=shop.payment_default_id.id
    return {'value':v}
```

When editing the shop_id form field, the onchange_shop_id method of the sale_order object is called and returns a dictionary where the 'value' key contains a dictionary of the new value to use in the 'project_id', 'pricelist_id' and 'payment_default_id' fields.

Note that it is possible to change more than just the values of fields. For example, it is possible to change the value of some fields and the domain of other fields by returning a value of the form: return {'domain': d, 'value': value} context in <record model="ir.actions.act_window" id="a"> you can add a context field, which will be pass to the action.

See the example below:

```
<record model="ir.actions.act_window" id="a">
  <field name="name">account.account.tree</field>
  <field name="res_model">account.account</field>
  <field name="view_type">tree</field>
  <field name="view_mode">form,tree</field>
  <field name="view_id" ref="v"/>
  <field name="domain">[( 'code', '=', '0')]</field>
  <field name="context">{'project_id': active_id}</field>
</record>
```

view_type:

tree = (tree with shortcuts at the left), form = (switchable view form/list)

view_mode:

tree,form : sequences of the views when switching

5.6 取得缺省值

Description

Get back the value by default for one or several fields.

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

Parameters:

- `fields`: the fields list which we want to recover the value by default.
- `form` (optional): TODO
- `reference` (optional): TODO

Returns: dictionary of the default values of the form `{'field_name': value, ... }`

Example:

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

default_set

Description

Change the default value for one or several fields.

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

Parameters:

- `field`: the name of the field that we want to change the value by default.
- `value`: the value by default.
- `for_user` (optional): boolean that determines if the new default value must be available only for the current user or for all users.

Returns: True

6 菜单和动作 (MENU 和 ACTION)

6.1 菜单(MENUS)

菜单定义格式:

```
<menuitem id="menuitem_id" name="Position/Of/The/Menu/Item/In/The/Tree" action="action_id"
icon="NAME_FROM_LIST" groups="groupname" sequence="<integer>" parent = "Parent Menu ID" />
```

其中

- **id**: 菜单ID, 唯一标志该菜单, ID不可以重复。该字段是必须的。
- **name**: 菜单显示名, 即界面上看到的菜单名, 该字段是必须的。可用符号"/"定义菜单的显示路径, 如 name="Sales Management/Sales Order/Sales Order in Progress", 表示销售管理模块下的菜单Sales Order下的子菜单Sales Order in Progress。如果指定了菜单路径, 可以不定义parent字段 (该字段定义菜单的上级菜单)。
- **action**: 菜单的响应动作的ID, 即点击该菜单, 系统调用哪个响应动作 (Action)。该字段不是必须的, 不指定 Action的菜单通常是父菜单。
- **icon**: 菜单的图标, 不是必须字段。默认icon是STOCK_OPEN。可用的icon有: STOCK_ABOUT, STOCK_ADD, STOCK_APPLY, STOCK_BOLD, STOCK_CANCEL, STOCK_CDROM, STOCK_CLEAR, STOCK_CLOSE, STOCK_COLOR_PICKER, STOCK_CONNECT, STOCK_CONVERT, STOCK_COPY, STOCK_CUT, STOCK_DELETE, STOCK_DIALOG_AUTHENTICATION, STOCK_DIALOG_ERROR, STOCK_DIALOG_INFO, STOCK_DIALOG_QUESTION, STOCK_DIALOG_WARNING, STOCK_DIRECTORY, STOCK_DISCONNECT, STOCK_DND, STOCK_DND_MULTIPLE, STOCK_EDIT, STOCK_EXECUTE, STOCK_FILE, STOCK_FIND, STOCK_FIND_AND_REPLACE, STOCK_FLOPPY, STOCK_GOTO_BOTTOM, STOCK_GOTO_FIRST, STOCK_GOTO_LAST, STOCK_GOTO_TOP, STOCK_GO_BACK, STOCK_GO_DOWN, STOCK_GO_FORWARD, STOCK_GO_UP, STOCK_HARDDISK, STOCK_HELP, STOCK_HOME, STOCK_INDENT, STOCK_INDEX, STOCK_ITALIC, STOCK_JUMP_TO, STOCK_JUSTIFY_CENTER, STOCK_JUSTIFY_FILL, STOCK_JUSTIFY_LEFT, STOCK_JUSTIFY_RIGHT, STOCK_MEDIA_FORWARD, STOCK_MEDIA_NEXT, STOCK_MEDIA_PAUSE, STOCK_MEDIA_PLAY, STOCK_MEDIA_PREVIOUS, STOCK_MEDIA_RECORD, STOCK_MEDIA_REWIND, STOCK_MEDIA_STOP, STOCK_MISSING_IMAGE, STOCK_NETWORK, STOCK_NEW, STOCK_NO, STOCK_OK, STOCK_OPEN, STOCK_PASTE, STOCK_PREFERENCES, STOCK_PRINT, STOCK_PRINT_PREVIEW, STOCK_PROPERTIES, STOCK_QUIT, STOCK_REDO, STOCK_REFRESH, STOCK_REMOVE, STOCK_REVERT_TO_SAVED, STOCK_SAVE, STOCK_SAVE_AS, STOCK_SELECT_COLOR, STOCK_SELECT_FONT, STOCK_SORT_ASCENDING, STOCK_SORT_DESCENDING, STOCK_SPELL_CHECK, STOCK_STOP, STOCK_STRIKETHROUGH, STOCK_UNDELETE, STOCK_UNDERLINE, STOCK_UNDO, STOCK_UNINDENT, STOCK_YES, STOCK_ZOOM_100, STOCK_ZOOM_FIT, STOCK_ZOOM_IN, STOCK_ZOOM_OUT, terppaccount, terpp-crm, terpp-mrp, terpp-product, terpp-purchase, terpp-sale, terpp-tools, terpp-administration, terpp-hr, terpp-partner, terpp-project, terpp-report, terpp-stock
- **groups**: 指定哪些权限组可以看见该菜单, 示例 (groups="admin,user")
- **sequence**: 菜单的显示序号, 序号越小, 菜单显示越靠上。默认值是10。

示例

在代码文件: server/bin/addons/sale/sale_view.xml, 有如下菜单定义。

```
<menuitem name="Sales Management/Sales Order/Sales Order in Progress" id="menu_action_order_tree4" action="
```

6.2 动作(ACTION)

概述

动作 (Action) 定义系统如何响应用户的操作。例如, 用户登录系统, 双击invoice, 点击菜单, 点击按钮, 等等。这些用户操作, 系统会调用相应的Action响应用户。

Action有多种类型:

- Window: 打开一个窗口
- Report: 打印一个报表, 包括 Custom Report, RML Report
- Wizard: 调用Wizard
- Execute: 执行Server端的一个方法(method)
- Group: 动作组, 即连续调用多个actions, 形成“动作串”。

动作常用场合有:

- 用户登录系统(User connection)
- 用户点击菜单
- 用户点击画面右边工具条上的Print或Action 中的链接。

事件例子

点击菜单

当用户点击菜单“Operations > Partners > Partners Contact”, 系统将发生如下响应:

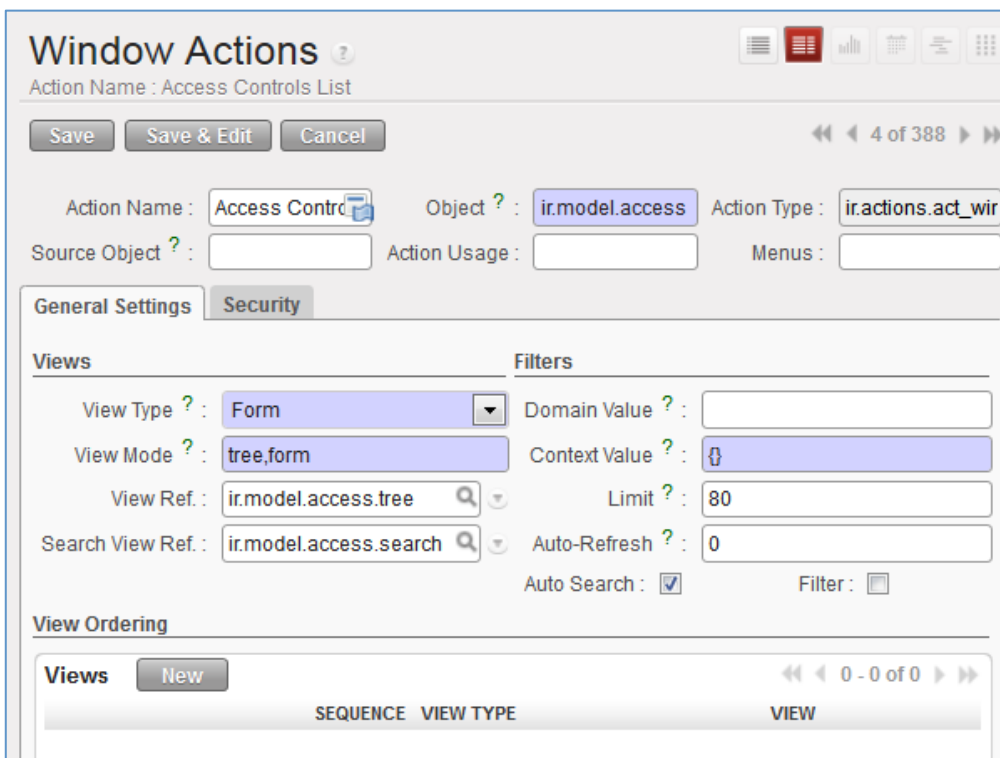
1. 在IR_相关表中查找对应Action
2. 执行Action: 如果Action的类型是Window, 系统打开一个新窗口 (List or Form), 显示选定的对象记录。

用户登录

当用户登录系统, 系统查找该用户的默认登录动作, 为用户显示特定画面。

1. 查找user file 取得 ACTION_ID
2. 执行Action

动作 (Action) 定义



Action Name: 动作名称, 可以任意取名。

Action Type: 动作类型, 总是 ‘ir.actions.act_window’

View Ref: 该动作关联的视图, 即动作应显示的畫面。

Object: 该动作关联的对象, 即动作应显示哪个对象的数据, 或者说应显示哪个数据表的数据。

Source Object: 指定应在哪个对象的视图的右边工具条上显示本Action。

view

The view describes how the edition form or the data tree/list appear on screen. The views can be of 'Form' or 'Tree' type, according to whether they represent a form for the edition or a list/tree for global data viewing. A form can be called by an action opening in 'Tree' mode. The form view is generally opened from the list mode (like if the user pushes on 'switch view').

domain

This parameter allows you to regulate which resources are visible in a selected view.(restriction)

For example, in the invoice case, you can define an action that opens a view that shows only invoices not paid.

The domains are written in python; list of tuples. The tuples have three elements;

- the field on which the test must be done
- the operator used for the test (<, >, =, like)
- the tested value

For example, if you want to obtain only 'Draft' invoice, use the following domain; [('state', '=', 'draft')]

In the case of a simple view, the domain define the resources which are the roots of the tree. The other resources, even if they are not from a part of the domain will be posted if the user develop the branches of the tree.

Window Action

Actions are explained in more detail in section "Administration Modules - Actions". Here's the template of an action XML record :

```
<record model="ir.actions.act_window" id="action_id_1">
  <field name="name">action.name</field>
  <field name="view_id" ref="view_id_1"/>
  <field name="domain">["list of 3-tuples (max 250 characters)"]</field>
  <field name="context">{"context dictionary (max 250 characters)"}</field>
  <field name="res_model">Open.object</field>
  <field name="view_type">form|tree</field>
  <field name="view_mode">form,tree|tree,form|form|tree</field>
  <field name="usage">menu</field>
  <field name="target">new</field>
</record>
```

Where

- id is the identifier of the action in the table "ir.actions.act_window". It must be unique.
 - name is the name of the action (mandatory).
 - view_id is the name of the view to display when the action is activated. If this field is not defined, the view of a kind (list or form) associated to the object res_model with the highest priority field is used (if two views have the same priority, the first defined view of a kind is used).
 - domain is a list of constraints used to refine the results of a selection, and hence to get less records displayed in the view. Constraints of the list are linked together with an AND clause : a record of the table will be displayed in the view only if all the constraints are satisfied.
 - context is the context dictionary which will be visible in the view that will be opened when the action is activated. Context dictionaries are declared with the same syntax as Python dictionaries in the XML file. For more information about context dictionaries, see section "The context Dictionary".
 - res_model is the name of the object on which the action operates.
 - view_type is set to form when the action must open a new form view, and is set to tree when the action must open a new tree view.
 - view_mode is only considered if view_type is form, and ignored otherwise. The four possibilities are : – form,tree : the view is first displayed as a form, the list view can be displayed by clicking the "alternate view button" ; – tree,form : the view is first displayed as a list, the form view can be displayed by clicking the "alternate view button" ; – form : the view is displayed as a form and there is no way to switch to list view ; – tree : the view is displayed as a list and there is no way to switch to form view.
- (version 5 introduced graph and calendar views)
- usage is used [+ *TODO* +]
 - target the view will open in new window like wizard.

They indicate at the user that he has to open a new window in a new 'tab'.

Administration > Custom > Low Level > Base > Action > Window Actions

Examples of actions

This action is declared in server/bin/addons/project/project_view.xml.

```
<record model="ir.actions.act_window" id="open_view_my_project">
  <field name="name">project.project</field>
  <field name="res_model">project.project</field>
  <field name="view_type">tree</field>
  <field name="domain">[(('parent_id', '=', False), ('manager', '=', uid))]</field>
  <field name="view_id" ref="view_my_project" />
</record>
```

This action is declared in server/bin/addons/stock/stock_view.xml.

```
<record model="ir.actions.act_window" id="action_picking_form">
  <field name="name">stock.picking</field>
  <field name="res_model">stock.picking</field>
  <field name="type">ir.actions.act_window</field>
  <field name="view_type">form</field>
  <field name="view_id" ref="view_picking_form"/>
  <field name="context">{'contact_display': 'partner'}</field>
</record>
```

Url Action

Wizard Action

Here's an example of a .XML file that declares a wizard.

```
<?xml version="1.0"?>
<terp>
<data>
  <wizard string="Employee Info" model="hr.employee"
    name="employee.info.wizard" id="wizard_employee_info"/>
</data>
</terp>
```

A wizard is declared using a wizard tag. See “Add A New Wizard” for more information about wizard XML. also you can add wizard in menu using following xml entry

```
<?xml version="1.0"?>
<terp>
<data>
<wizard string="Employee Info" model="hr.employee" name="employee.info.wizard" id="wizard_employee_info"/>
<menuitem name="Human Resource/Employee Info" action="wizard_employee_info" type="wizard"
id="menu_wizard_employee_info"/>
</data>
</terp>
```

Report Action

Report declaration

Reports in Open ERP are explained in chapter “Reports Reporting”. Here's an example of a XML file that declares a RML report :

```
<?xml version="1.0"?>
<terp>
<data>
<report id="sale_category_print" string="Sales Orders By Categories" model="sale.order"
name="sale_category.print" rml="sale_category/report/sale_category_report.rml" menu="True" auto="False"/>
</data>
</terp>
```

A report is declared using a report tag inside a “data” block. The different arguments of a report tag are :

- id : an identifier which must be unique.
- string : the text of the menu that calls the report (if any, see below).

-
- model : the Open ERP object on which the report will be rendered.
 - rml : the .RML report model. Important Note : Path is relative to addons/ directory.
 - menu : whether the report will be able to be called directly via the client or not. Setting menu to False is useful in case of reports called by wizards.
 - auto : determines if the .RML file must be parsed using the default parser or not. Using a custom parser allows you to define additional functions to your report.

7 OPENERP 工作流

7.1 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 OpenERP 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。

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)系统自动产生收货单，工仓库收货；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。

7.2 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，即执行本节点的动作。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，才执行本节点的动作。

flow_start: 表示流程的开始节点。

flow_stop: 表示流程的结束节点。

7.3 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="group_id" ref="groupid"/>
  <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。

group_id: 权限组，表示只有该权限组可以触发本迁移。

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， workflow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。

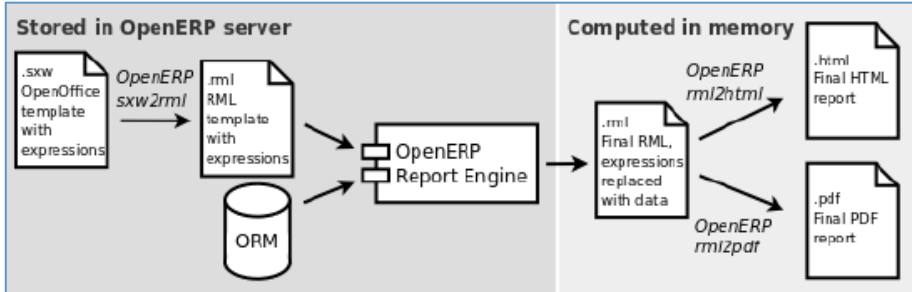
condition: 迁移的条件，是一段 Python 代码，通常是一个函数调用。当系统收到 **signal** 中定义的信号时候，检查此处的条件，条件为真则实际触发迁移。

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>
```


8 OPENERP 报表开发

8.1 OpenERP 报表运行机制



1) OpenERP 报表的基本运行机制

OpenERP 报表的一般定义语法是：

```
<report id="c2c_demo_report_x" string="C2C Demo Report" model="hr.holidays"
name="sandbox_c2c_reporting_tools" auto="False" header="False"/>
```

这个定义的含义是，在对象 `hr.holidays` 上增加报表操作（`model="hr.holidays"`），该报表操作的显示字符是 `C2C Demo Report`（`string="C2C Demo Report"`），当用户点击该操作字符（`C2C Demo Report`），系统调用名为 `sandbox_c2c_reporting_tools`（`name="sandbox_c2c_reporting_tools"`）的 `Services`，该 `Services` 返回报表文件（PDF 或其他格式文件）。

因此，理解 OpenERP 报表机制的核心是，理解报表 `Services` 机制。

2) OpenERP 的报表 Service

OpenERP 的报表 `Service` 的基本接口定义在文件：`openerp-server-6.0.3\bin\report\interface.py`，期定义如下：

```
report_int(netsvc.Service)
__init__(self, name, audience='')
create(self, cr, uid, ids, datas, context=None)
```

`init` 方法中最重要的参数是 `name`，该参数是 `Service Name`，其格式是 `"report.xxx"`，`xxx` 必须和报表定义时候的（`name="sandbox_c2c_reporting_tools"`）一致，系统是通过该名字找到该 `Service`。

`create` 方法中，最重要的参数是 `ids`，该参数是报表操作所在的画面上，选定的对象的 `id` 列表。通常，系统会为 `ids` 中的每一个对象出一个报表。`datas` 参数通常用于 `Wizard` 的情况，即先弹出 `Wizard` 画面，用户输入一些数据，点击按钮，系统再输出报表文件。在这种情况下，`datas` 参数里保存着用户在 `Wizard` 画面上输入的数据。

显然，系统的内部动作是，用户点击报表动作，系统根据 `name="sandbox_c2c_reporting_tools"` 找到相应 `Service`，调用 `Service` 的 `Create` 方法，返回报表文件。`Create` 方法的返回值格式是：`(report_doc,mimetype)`。例如，如果返回 pdf 报表，返回值是 `(pdf_doc,'pdf')`。

3) RML 报表

如果直接继承接口 `report_int`，编写 `create` 方法生成 pdf 文档，代码复杂，工作量大。系统提供了 `RML` 格式报表，用于简化 pdf 报表开发。其基本原理是，开发 `RML` 格式文档，系统的 `Create` 方法读取 `rml` 文件，渲染成 pdf 文档，输出。相关接口如下：

```
report_rml(report_int)
__init__(self, name, table, tmpl, xsl)
create(self, cr, uid, ids, datas, context)
```

```
report_sxw(report_rml)
__init__(self, name, table, rml=False, parser=rml_parse, header='external', store=False)
create(self, cr, uid, ids, data, context=None)
```

这两个派生 Class 中，**create** 方法的参数没有变化，**init** 方法增加了一些参数，说明如下：

- ◆ **table**: 报表关联的数据对象，渲染 rml 时候需要调用该对象取得数据。
- ◆ **rml**: RML 文件路径及名称，系统需要读取该文件渲染成 PDF 报表。
- ◆ **parser**: 渲染器，系统的实际做法是，在 **create** 方法中调用渲染器的有关方法，将 rml 渲染成 pdf。

用户可以开发自己的渲染器，用于将 rml 渲染成其他格式，如 html、txt 等，实际上，系统已经提供了 html、txt 等的渲染器。因此，开发 rml 格式的报表时候，通常只需要开发自己的渲染器（**parser**），不需要开发 **report_int**。

8.2 RML 报表开发方法

8.2.1 概述

OpenERP 系统提供了 RML 格式报表，用于简化 pdf 报表开发。其基本原理是，开发 RML 格式文档，系统的 **Create** 方法读取 rml 文件，渲染成 pdf 文档，输出。

添加 **report** 的定义格式：

```
<report id="sale_category_print"
string="Sales Orders By Categories"
model="sale.order"
name="sale_category.print"
rml="sale_category/report/sale_category_report.rml"
menu="True"
auto="False"/>
```

- ◆ **id** : 任意字符串，不重复即可。
- ◆ **string**:
- ◆ **model**: 报表渲染对象
- ◆ **rml**: rml 文件路径，路径以本模块的路径开始
- ◆ **menu**: True or False，是否在客户端上显示菜单，**False** 一般用语 Wizard 向导是，不需要菜单。
- ◆ **auto**: 是否利用缺省方法分析 .RML 文件
- ◆ **name**: 报表名称（**report_sxw.report_sxw** 中的 第一个参数 去掉 ‘report.’）

8.2.2 RML 语法格式

```
<document>
  <!--对报表页面做设置-->
  <template pageSize="21cm,29.7cm">
    <pageTemplate>
```

```
<frame id="first" x1="2.0cm" y1="2.5cm" width="17cm" height="25.0cm"/>
</pageTemplate>
</template>
<!--对报表显示页面样式做定义-->
<stylesheet>
  <paraStyle name="Standard" fontName="Helvetica" fontSize="14.0" leading="16.0"
    alignment="CENTER"/>
  <paraStyle name="P01" fontName="Helvetica" spaceBefore="0.0" spaceAfter="6.0"
    fontSize="14.0"/>
  <blockTableStyle id="Table01">
    <blockFont name="Helvetica" size="14.0" start="0,0" stop="-1,-1"/>
    <blockAlignment start="0,0" stop="-1,-1" value="CENTER"/>
    <blockAlignment start="0,0" stop="1,2" value="LEFT"/>
    <blockValign start="0,0" stop="-1,-1" value="MIDDLE"/>
    <lineStyle kind="GRID" colorName="black" start="0,0" stop="-1,-1"/>
  </block>
</stylesheet>
<!--正文-->
<story>
  <!--引用全局对象 objects-->
  <para>[[repeatIn(objects,'O')]]</para>
  <!--引用样式 style="Standard"-->
  <para style="Standard">介绍</para>
  <!--表格内容，colwidths 设置列宽度 rowHeights 设置行高度-->
  <blockTable colWidths="2.2cm,5.9cm,2.2cm,6.3cm"
    rowHeights="1.0cm,1.0cm,3.0cm,1.0cm" style="Table01">
    <tr>
      <td><para style="Standard">姓名</para></td>
      <td>[[ o.name ]]</td>
    </tr>
  </blockTable>
</story>
</document>
```

8.2.3 Report Action 配置

1. OpenOffice 添加插件 工具→→扩展管理器 openerp_report_designer.zip

2. 安装 OpenERP 模块 base_report_designer

3. 创建模块并放入 OpenERP \Server\addons\ 目录下

3.1 搭建模块文件

3.2 编写报表打印视图 xml 文件

例如: <report id="report_qingjia_rpt_qingjd" model="qingjia.qingjd" name="qingjia.rpt_qingjd" rml="qingjia/rpt_qingjd.rml" string="打印请假单"/>

id: 报表 ID

model: 所引用的对象且定义该报表打印按钮显示位置

name: 报表的 name (打印报表跳转关键所在)

rml: 规范写法 “模块名/rml 文件” (写出 rml 文件的相对路径, 不固定)

string: 打印报表按钮显示名称

8.2.4 Parser

例如:

```
import time
from report import report_sxw
from osv import osv

class rpt_qingjd(report_sxw.rml_parse):
    def __init__(self, cr, uid, name, context):
        super(rpt_qingjd, self).__init__(cr, uid, name, context)
        self.localcontext.update({
            'time': time,
        })
    report_sxw.report_sxw('report.qingjia.rpt_qingjd','qingjia.qingjd',
        'addons/qingjia/rpt_qingjd.rml',parser=rpt_qingjd)
```

Class rpt_qingjd: 为报表的语法分析类，报表中定义的方法源于此类

“report.qingjia.rpt_qingjd”:为报表名称，规范写法“report.***”，“***”对应报表 xml 文件中的 name

“qingjia.qingjd”: 报表关联的数据对象，渲染 rml 时候需要调用该对象取得数据。

“addons/qingjia/rpt_qingjd.rml”: RML 文件路径及名称，系统需要读取该文件渲染成 PDF 报表。

Parser=rpt_qingjd: 渲染器，系统的实际做法是，在 create 方法中调用渲染器的有关方法，将 rml 渲染成 pdf。

8.3 Aeroo 报表开发方法

8.3.1 概述

一个基于 OpenOffice 为 OpenERP 实现全面的报表引擎，支持多种格式，简单灵活。使用 Aeroo 报表引擎，可以开发出 Excel、Word、Excel 等各种格式的、非常漂亮的报表。关于 aeroo 引擎的安装方法等，可以参考这里：http://blog.sina.com.cn/s/blog_700b28cc0100wgsr.html。

8.3.2 Aeroo 语法格式

Odt: < %> (% 为 python 代码)

OpenOffice 操作: 插入→字段→功能→输入字段 可进行设置

Ods: python://% (% 为 python 代码)

OpenOffice 操作: 插入→超链接 可进行设置

8.3.3 AerooAction 配置

1. 安装 OpenOffice

2. 创建模块并放入 OpenERP \Server\addons\ 目录下

2.1. 搭建模块文件

2.2. 编写 Aeroo Report xml 文件

```
<record model='ir.actions.report.xml' id='ir_actions_report_xml_printscreen_list'>
```

```
<!--定义 Aeroo Report type-->
<field name='report_type'>aeroo</field>
<!--定义 Aeroo Report 初始化文件格式-->
<field name='in_format'>oo-ods</field>
<!--定义 Aeroo Report 最后显示文件格式-->
<field name='out_format' eval="ref('report_aeroo.report_mimetypes_ods_ods')"/>
<!-->
<field name='multi' eval='False'/>
<!--定义显示报表名称-->
<field name='name'>Printscreen List</field>
<!--定义 Aeroo Report 文件-->
<field name='report_rml'>report_aeroo_printscreen/data/template.ods</field>
<!--定义报表 name 属性-->
<field name='report_name'>printscreen.list</field>
<!--定义报表中所引用的 objects-->
<field name='model'>ir.ui.view</field>
<!--定义报表文件在数据库中的 type-->
<field name='type'>ir.actions.report.xml</field>
<!--定义使用附件-->
<field name='attachment_use' eval='False'/>
<!--定义 Parser,即报表所对应的 py 文件-->
<field name='parser_loc'>report_aeroo_printscreen/parser.py</field>
<!-->
<field name='tml_source'>file</field>
<!--定义报表所对应的 py 文件状态-->
<field name='parser_state'>loc</field>
<!--定义样式-->
<field name='styles_mode'>default</field>
</record>
```

8.3.4 Parser

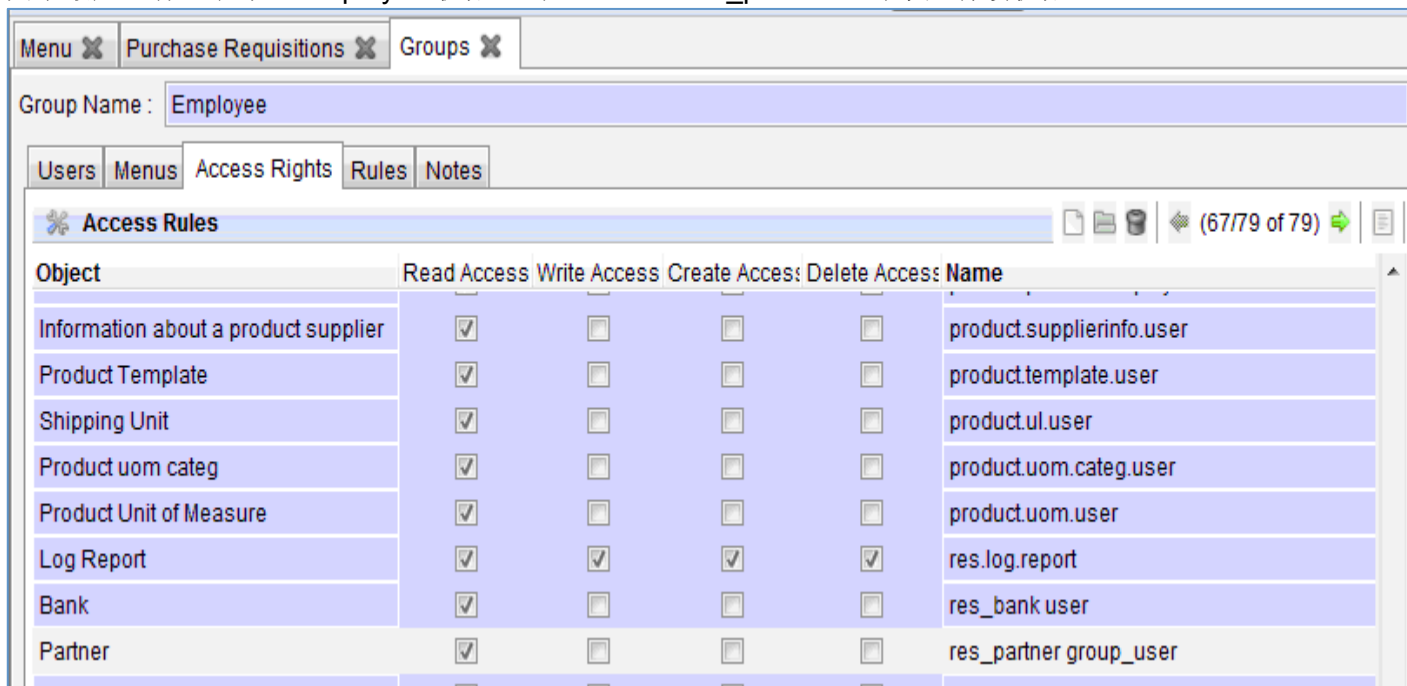
```
import time
from report import report_sxw
class Parser(report_sxw.rml_parse):
    def __init__(self, cr, uid, name, context):
        super(Parser, self).__init__(cr, uid, name, context)
        self.localcontext.update({
            'time':time,
        })
```

Class Parser: 为报表的语法分析类，报表中定义的方法源于此类(注意：此类名不能更改)

9 OPENERP 权限设置

OpenERP 的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。



The screenshot shows the 'Access Rights' configuration window for the 'Employee' group. The 'Access Rules' tab is active, displaying a table of permissions for various objects. The last row shows that the 'Employee' group has 'Read' access to the 'Partner' object (res_partner group_user).

Object	Read Access	Write Access	Create Access	Delete Access	Name
Information about a product supplier	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	product.supplierinfo.user
Product Template	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	product.template.user
Shipping Unit	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	product.ul.user
Product uom categ	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	product.uom.categ.user
Product Unit of Measure	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	product.uom.user
Log Report	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	res.log.report
Bank	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	res_bank user
Partner	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	res_partner group_user

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。

Menu	Purchase Requisitions	Groups	Record Rules
------	-----------------------	--------	--------------

General

Access Rights

Name:
Apply For Read: ☐
Apply For Write: ☒

Object:
Apply For Create: ☒
Apply For Delete: ☒

Domain Setup

Domain:

Groups (no group = global)

Global: ☐

Group Name
Employee

如上图表示，Employee 权限组，对申购单（Purchase Requisition）对象中，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，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。
`<field name="variants" groups="product.group_product_variant"/>`

workflow 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：`<field name="group_id" ref="groupid"/>`。

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。

10 OPENERP WEB 开发

10.1 Web 运行机制

1) OpenERP Web 框架原理

1.1) 系统启动

OpenERP 系统启动代码见 `openerp/cli/server.py`，系统启动时候，首先检查多核参数（`workers`），如果是多进程模式（`multi_process`），则启动多核版（`Multicorn`）`WSGI_Server`，否则启动单核版 `WSGI_Server`。如果是单进程模式，系统同时启动 `netrpc_server` 和 `cron` 线程。多进程模式，则不启动 `netrpc_server` 和 `cron` 线程，相关代码参见 `openerp/service/__init__.py`。因此，多进程模式下，基于 `netrpc` 的 GUI 客户端不能用，同时，系统正常运行必不可少的 `cron` 作业需要另开进程启动。

1.2) WSGI_Server 启动

`WSGI_Server` 启动后，侦听 `http` 请求。当收到请求后，轮询 `handler`，如果某 `handler` 返回了结果，则返回给客户。如果没有合适的 `handler` 响应，则返回 `404` 错误页面。

`handler` 分系统 `handler` 和模块 `handler`，系统 `handler` 是系统启动时候加载的，模块 `handler` 是，加载含有 `handler` 的模块时候，注册 `handler` 到 `WSGI_Server`。系统 `handler` 有：`wsgi_xmlrpc_1`，`wsgi_xmlrpc`，`wsgi_xmlrpc_legacy`，`wsgi_webdav`，他们处理下述路径的 `web` 请求：

```
XML_RPC_PATH = '/openerp/xmlrpc'
XML_RPC_PATH_1 = '/openerp/xmlrpc/1'
JSON_RPC_PATH = '/openerp/jsonrpc'
JSON_RPC_PATH_1 = '/openerp/jsonrpc/1'
```

相关代码见：`wsgi_server.py` 的 `application_unproxied`。

系统目前只有一个模块 `handler`，即 `addons/web` 模块提供的 `handler`。该模块加载时，即注册 `handler`。该 `handler` 中，收到 `http` 请求时候，根据请求的 `path`，查找 `http_controller`，找到 `controller` 则调用该 `controller` 的方法。

`http_controller` 的原理是，OpenERP 系统将 `addons` 下含有 `static` 目录的模块当成 `http_controller`。系统加载该模块时候，自动将模块中的继承自 `web.http.Controller` 的 `class` 当作 `http_controller`。每个 `http_controller` 有属性 `_cp_path`，表明该 `controller` 处理 `_cp_path` 下的请求。系统根据请求路径比对 `_cp_path`，找到相应 `controller`，调用 `controller` 的 `method` 响应请求。请求路径中最后一个 `/` 之后的内容，系统默认为是 `controller` 的 `method` 名。上述相关代码见：`openerp/addons/web/http.py`。

2) Web 模块开发

OpenERP `web` 模块很简单，只要在普通模块中，a)增加目录 `static`，其中通常存放静态网页内容，包括 `js`，`img`，`xml` 模板等。b)通常增加一个 `controllers` 目录(不是必须的)，里面放 `py` 的 `http_controller` 代码。每个 `controller` 都必须继承自 `openerp.addons.web.http.Controller`，并且必须有属性 `_cp_path`。`controller` 中的每个 `method` 相应一种 `http` 请求。

10.2 Web 页面开发

参考例子：`web_hello`，`point_of_sale`