[QWE](http://odoodevelop.lofter.com/post/1cd05c40_41e3faa)B

**1.web 模块**

注意，OpenERP 模块中 web 部分用到的所有文件必须被放置在模块内的 static 文件夹里。这是强制性的，出于安全考虑。

事实上，我们创建的文件夹 CSS，JS 和 XML，仅仅是一个习惯。

static文件夹

oepetstore/static/css/petstore.css 是我们的 CSS 文件。

oepetstore/static/xml/petstore.xml 是一个 XML 文件，将包含我们 QWeb 的模板。

oepetstore/static/js/petstore.js包含应用程序的JavaScript 代码。

像 OpenERP 的 XML 文件包含了视图或数据一样，必须在\_\_openerp\_\_.py文件内标明这些文件。下面是我们增加的行，它告诉 web client 必须记载这些文件：

'js': ['static/src/js/\*.js'],

'css': ['static/src/css/\*.css'],

'qweb': ['static/src/xml/\*.xml'],

OpenERP内，默认会把所有的 JavaScript 文件连接为一个文件。然后，我们执行一个叫 minification的操作。

minification 将移除文件中的所有的注释、空格和换行符。最后，发送这个文件给用户浏览器。

但这么做的缺点，是无法调试应用程序。为了避免这种副作用，仍然能够调试的解决办法是：在 OpenERP 的 URL 后面添加一个参数 ?debug .

添加后的URL：<http://localhost:8069/?debug>

当您使用带 debug 参数的 URL，应用程序将不会执行串联所有 minification的 JavaScript 文件这个过程。应用程序也将需要更多的时间来加载，但你能进

行调试开发了。

在前面的章节中，我们解释了 JavaScript 缺少命名空间机制，来分割在不同的 JavaScript 文件中声明的变量。并且我们提出了模块模式这个简单方法。

模块模式（如 app.js文件）：

(function() {

app = {};

function main() {

console.log("launch application");

};

app.main = main;

})();

在 OpenERP 的 web 框架内，有个类似于模块模式的等价物，集成了该框架的其余部分。请注意，OpenERP 的 Web 模块与其他 OpenERP 的 addon 模块概念上是不同的，一个 addon 模块是一个包含很多文件的文件夹，web 模块仅仅是一个有命名空间概念的 JavaScript。

**1)模块**

oepetstore/static/js/petstore.js 声明了这样的模块:

**openerp.oepetstore**=function(instance){

instance.oepetstore={};

instance.oepetstore.XXX=......;

}

在OpenERP的Web框架内，通过声明一个函数来声明一个 JavaScript 模块，并把这个函数放在全局变量openerp的属性内.这个属性名称必须和OpenERP addon 模块名称一致 （这 addon 模块名为 oepetstore，我应把函数赋值给openerp.oepetstore属性。如果换成openerp.petstore属性，则无法正常运行）。当 Web 客户端转载这个 addon 模块时，该函数将被调用。将传入一个名为instance的参数，这个参数代表当前 OpenERP 的 Web 客户端实例， 包含了所有相关当前会话数据，以及所有 Web 模块的变量。

在 instance 对象内创建与 addon 模块名称一致的新命名空间是个惯例。这就是为什么我们在 instance.oepetstore 设置一个空 dictionary。这个 dictionary就是命名空间，用来声明我们模块内自己使用的所有类和变量。

**2）类**

JavaScript 不像其他面向对象编程语言那样有类机制。更确切地说，它提供了面向对象编程语言元素，但你必须自己定义，自己选择如何做。

OpenERP Web 框架提供工具来简化这个过程，让程序员以类似其他编程语言，如 Java 的方式编码。

定义一个新类，你需要从 instance.web.Class 类继承。语法如下：

instance.oepetstore.MyClass = instance.web.Class.extend({

say\_hello: function() {

console.log("hello");

},

});

类的实例化：

var my\_object = new instance.oepetstore.MyClass();

my\_object.say\_hello();

类可以有一个构造函数，它方法名为 init()。你可以像大多数开发语言那样，传递参数给构造器：

instance.oepetstore.MyClass = instance.web.Class.extend({

init: function(name) {

this.name = name;

},

say\_hello: function() {

console.log("hello", this.name);

},

});

类的继承

类可以被继承

instance.oepetstore.MySpanishClass = instance.oepetstore.MyClass.extend({

say\_hello: function() {

this.\_super();

console.log("hola", this.name);

},

});

var my\_object = new instance.oepetstore.MySpanishClass("Nicolas");

my\_object.say\_hello();

this.\_super()不是常用的类方法

**3）Widgets Basics（基础部件）**

OpenERP中Widget部件，是一个通用组件，专门用来向用户显示内容。

oepetstore实例中的petstore.js内容：

openerp.oepetstore=function(instance){

var \_t=instance.web.\_t,

\_lt=instance.web.\_lt;

var QWeb=instance.web.qweb;

instance.oepetstore={};

<!--部件-->

instance.oepetstore.HomePage=instance.web.Widget.extend({

start:function(){

console.log("pet store home page loaded");

},

});

instance.web.client\_actions.add('petstore.homepage','instance.oepetstore.HomePage');

}

最后一行代码，把这个部件注册为客户端的action。当我们点击‣Pet Store ‣ Pet Store ‣ Home Page菜单项时，客户端action让部件显示出来。

HomePage 部件有一个 start() 方法。 在部件初始化后，这方法被自动调用。它已接接受指令去显示其内容。我们将用它向用户显示一些内容。要做到这一点，我们使用所有部件都有的 $el 属性。该属性是一个 jQuery对象， 表示部件对应的 HTML 标签的根标签。部件包含了多个 HTML 标签，这些 HTM 标签有一个统一的根标签。默认情况下，部件都有一个空的根标签：一个<div>。

一个<div> HTML 标签在没有具体内容的时候，是不可见的。这也解释了为什么显示 instance.oepetstore.HomePage 时， 是个空白区域， 它根本没有任何内容。要想显示些内容，我们用 jQuery 该对象上的一些简单方法，在根标签中添加一些 HTML 标签：

instance.oepetstore.HomePage = instance.web.Widget.extend({

start: function() {

this.$el.append("<div>Hello dear OpenERP user!</div>");

},

});

**常用方法：**

appendTo()方法

var pettoys = new instance.oepetstore.PetToysList(this);//实例化PetToysList()部件

pettoys.appendTo(this.$(".oe\_petstore\_homepage\_left"));//把pettoys添加到当前部件的class=.oe\_petstore\_homepage\_left的标签中

注意：new instance.oepetstore.PetToysList(this)，其中参数this，代表调用此部件的实例，表示部件的隶属关系。

addClass( )方法

this.$el.addClass("oe\_petstore\_homepage");//添加样式

getParent()方法

用于获取父部件

getChildren() 方法

用来获取子部件列表:this.getChildren()[0].$el

当你在部件中重载init()时，必须以父部件作为第一参数传入，并调用传入给this.\_super(),例如：

instance.oepetstore.GreetingsWidget = instance.web.Widget.extend({

init: function(parent, name) {

this.\_super(parent);

this.name = name;

},

});

最后，如果一个部件没有父部件（即：在你的应用实例化时是第一个部件），传入 null 参数：

new instance.oepetstore.GreetingsWidget(null);

**销毁部件：**

destroy()方法：

当一个部件被销毁，它会先调用所有子部件的destroy()。然后，它在DOM中清楚自己。通过部件隶属关系的递归调用，避免了内存泄漏，这对容易产生内

存泄露的大型JavaScript应用程序来说是非常有用的。

**2.Qweb**

在 OpenERP 中，使用Qweb模板引擎，专门用于Web 客户端开发。

Qweb 是一种基于XML的模板语言，类似Genshi，Thymeleaf 或 Facelets。

有如下几个特点：

① 在浏览器中完全用 JavaScript 执行。

② 每个模板文件（XML 文件）包含了多个模板，而其他模板引擎通常做法是，模板文件和模板之间 1:1 的关系。

③ 在 OpenERP 的 Web 的 instance.web.Widget 特别支持了 QWeb。虽然QWeb 可以用于 OpenERP 的 Web 客户端以外的地方（同时instance.web.Widget 也可以不依赖于 Qweb）。

之所以没用用其他 JavaScript 模板引擎，而是选择了 QWeb， 是因为 QWeb的扩展机制与 OpenERP 的视图继承机制很相似。就像 OpenERP 的视图一样，QWeb 模版也是个 XML 树结构，因此很容易在模版执行 XPath 或 DOM 操作。

**在部件内使用 QWeb**

首先，在文件 oepetstore/static/src/xml/petstore.xml 里我们定义一个简单的QWeb 模板。

<?xml version="1.0" encoding="UTF-8"?>   
<templates xml:space="preserve">

<t t-name="HomePageTemplate">

<div style="background-color: red;">This is some simple HTML</div>

</t>

</templates>   
现在，修改HomePage类：

var QWeb = instance.web.qweb;

我们建议在所有的 OpenERP web 模块复制粘贴这行代码。这个对象提供访问被Web客户端加载的所有模版文件中的模版的功能。下面是我们使用XML模板文件定义的模版的例子：

instance.oepetstore.HomePage = instance.web.Widget.extend({

start: function() {

this.$el.append(QWeb.render("HomePageTemplate"));

},

});

解释：QWeb.render()方法用来渲染指定标识名称的具体模版，第一个参数就是模版标识名称。

另外一个常用地方是，在部件内集成 Qweb：

instance.oepetstore.HomePage = instance.web.Widget.extend({

template: "HomePageTemplate",

start: function() {

...

},

});

当你在部件内设置类属性 template 时，部件就知道它需要调用 QWeb.render()来呈现该模板。

请注意这两个语法是有区别的。当在部件内集成 Qweb 时， QWeb.render() 调用在部件调用 start() 之前发生，并用模版的根标签替换了部件的默认根标签。这会导致不同的结果，所以你应该记住它。

**Qweb 上下文（Context）**

像所有的模板引擎一样， Qweb 模板可以包含操纵传递给模板的数据的代码。

1）使用QWeb.render()的第二个参数传递数据给模版：

<t t-name="HomePageTemplate">

<div>Hello <t t-esc="name"/></div>

</t>

QWeb.render("HomePageTemplate", {name: "Nicolas"});

输出结果： <div>Hello Nicolas</div>

2）当部件整合 QWeb 时，就不能直接传入数据了。 替换办法是，模板有一个唯一变量 widget ，这个变量引用了当前部件：

<t t-name="HomePageTemplate">

<div>Hello <t t-esc="name"/></div>

</t>

instance.oepetstore.HomePage = instance.web.Widget.extend({

template: "HomePageTemplate",

init: function(parent) {

this.\_super(parent);

this.name = "Nicolas";

},

start: function() {

},

});

输出结果： <div>Hello Nicolas</div>

**模版声明**

Qweb 的语法：

Qweb 指令使用前缀 t- 的 XML 属性来声明新模板，我们在 XML 文件的根元素 <template> 内添加一个 <t t-name="..."> 元素：

<templates>

<t t-name="HomePageTemplate">   
<div>This is some simple HTML</div>   
</t>

</templates>

**Escaping————转义**

用 t-esc :在 HTML 中放置文本

<t t-name="HomePageTemplate">

<div>Hello <t t-esc="name"/></div>

</t>

这将输出变量 name ，并转义变量的上下文，这上下文也许是类似 HTML的字符串。请注意，该属性 t-esc 可包含各种 JavaScript 表达式：

<t t-name="HomePageTemplate">

<div><t t-esc="3+5"/></div>

</t>

**输出HTML**

如果你明确知道变量会包含一些 HTML 标签，使用 t-raw 代替的 t-esc ：

<t t-name="HomePageTemplate">

<div><t t-raw="some\_html"/></div>

</t>

**条件**

QWeb 的条件关键字是 t-if ：

<t t-name="HomePageTemplate">

<div>

<t t-if="true == true">

true is true

</t>

<t t-if="true == false">

true is not true

</t>

</div>

</t>

Qweb 条件分支没有“else”这个分支结构。

**循环**

遍历列表，使用 t-foreach 和 t-as

<t t-name="HomePageTemplate">

<div>

<t t-foreach="names" t-as="name">

<div>

Hello <t t-esc="name"/>

</div>

</t>

</div>

</t>

**设置任意XML属性值**

Qweb 有一个特定语法设置属性值。您必须使用 t-att-xxx 并用属性名替换xxx：

<t t-name="HomePageTemplate">

<div>

Input your name:

<input type="text" t-att-value="defaultName"/>

</div>

</t>

**3.** **部件事件和属性**

**1）事件**

部件类似现有大部分图形用户界面库(Qt, GTK, Swing,...)那样，触发事件，

处理事件。例如：

instance.oepetstore.ConfirmWidget = instance.web.Widget.extend({

start: function() {

var self = this;

this.$el.append("<div>Are you sure you want to perform this action?</div>" +

"<button class='ok\_button'>Ok</button>" +

"<button class='cancel\_button'>Cancel</button>");

this.$el.find("button.ok\_button").click(function() {

self.trigger("user\_choose", true);

});

this.$el.find("button.cancel\_button").click(function() {

self.trigger("user\_choose", false);

});

},

});

instance.oepetstore.HomePage = instance.web.Widget.extend({

start: function() {

var widget = new instance.oepetstore.ConfirmWidget(this);

widget.on("user\_choose", this, this.user\_choose);

widget.appendTo(this.$el);

},

user\_choose: function(confirm) {

if (confirm) {

console.log("The user agreed to continue");

} else {

console.log("The user refused to continue");

}

},

});

注意：var self = this;请记住，在 JavaScript 中，变量 this 会隐含传递给所有函数。如果函数像对象引用方法那么用，通过 this 我们知道当前的对象是什么。每个已声明的函数都有自己的 this。所以，当我们在一个函数内声明了另一个函数，这个新功能将有自己的 this ，这和父函数 this 含义不同。如果我们要用原来的对象 this ，最简单的方法是把引用存储在一个本地变量。在 OpenERP 内，按照 Python 的习惯，通常该变量命名为 self。

由于这个部件应该是通用的，它本身不应执行任何具体 action。所以，我们用 trigger() 方法设置了名字叫"user\_choose"的触发器和事件。

Widget.trigger(event\_name [ , ...]) 方法的第一个参数是待触发的事件名，也接受任何数量的其他参数。这些参数将被传递到所有的事件侦听器。

然后，我们修改部件HomePage去实例化一个部件ConfirmWidget，通过on( )方法监听"user\_choose"事件。Widget.on(event\_name,object, func) 允许绑定一个事件 event\_name 触发时调用的函数func。如果func是个方法，则object是func函数的引用关联对象。当func被调用时，trigger()的其他参数会传递给它。例如：

start: function() {

var widget = ...

widget.on("my\_event", this, this.my\_event\_triggered);

widget.trigger("my\_event", 1, 2, 3);

},

my\_event\_triggered: function(a, b, c) {

console.log(a, b, c);

// will print "1 2 3"

}

**2）属性**

属性类似与普通对象的属性。可以在对象上设置数据，但有个额外的功能：属性值改变时将触发事件。

start: function() {

this.widget = ...

this.widget.on("change:name", this, this.name\_changed);

this.widget.set("name", "Nicolas");

},

name\_changed: function() {

console.log("The new value of the property 'name' is", this.widget.get("name"));

}

Widget.set(name,value) 方法设置某个属性的值。如果该值改变（或以前没有值），对象将触发一个事件change:xxx。xxx 是属性名称。

**4.部件辅助工具**

**1）部件的 jQeruy 选择器**

find() 方法

this.$el.find("input.my\_input")

$()方法

this.$("input.my\_input")

说明：我们强烈建议你也不要使用，全局 jQuery函数$()。这种全局选择器满足简单应用，但在真正的大型 web 应用程序中不好。原因很简单：当你创建一个新部件，你永远不知道它会实例化多少次。由于 $() 全局函数是操作浏览器中的全部 HTML，如果你实例化一个部件两次，该函数会搞混两个部件的个内容。这就是为什么，大部分时间里，你在定位部件里的 HTML 时，必须限制 jQuery 选择器的选择范围。

出于同样的逻辑，你也可以猜测到，不能够在部件里使用 HTML id。如果widget 被实例化的两次，在应用程序里将有两个相同 id，但却是不同的 HTML元素的情况。而这本身就是一个错误。所以， 在所有的情况下，你应该坚持使用用 CSS 类去标记 HTML 标签。

**2） 简易 DOM 事件绑定**

在前面的一部分，我们必须用 click() 或 change() 等事件绑定 HTML 元素。现在，我们用 $() 来简化代码，让我们看看如何做：

instance.oepetstore.MyWidget = instance.web.Widget.extend({

start: function() {

var self = this;

this.$(".my\_button").click(function() {

self.button\_clicked();

});

},

button\_clicked: function() {

..

},

});

还有有一个更简单的语法：

instance.oepetstore.MyWidget = instance.web.Widget.extend({

events: {

"click .my\_button": "button\_clicked",

},

button\_clicked: function() {

...

}

}); 把在DOM上触发的jQuery事件和部件的事件区分开很重要。event类属性是一个“助手”，协助绑定jQuery 的事件，它与部件事件无关，部件事件通过on()方法绑定。

event类属性是一个dictionary，**dictionary的key是空格隔开成两个部分的字符串**。第一部分是事件的名称，第二个是jQuery选择器。所以key click.my\_button将绑定在所有CSS类名为“my\_button”的HTML标签的 click事件上。**dictionary的value值是对象内被调用的方法名称**。

**5.开发指南**

遵循原则：

1）标识符（id属性），应尽量避免使用。在通用的应用程序和模块里，id限制了组件的可复用性，往往使代码更加脆弱。几乎所有时候，id可为空，使用 CSS类或保留一个DOM节点引用，或嵌套在一个jQuery的元素内去引用。

如果绝对必要使用id（因为第三方库需要，并且无法获取一个DOM元素），应该用\_.uniqueId()生成。

2）避免可预见/常见的CSS类名。CSS 类名，如content或navigation能与想要表达的意思/语义匹配。但其他开发人员将有同样想法，从而发生命名冲突和意外行为。例如，通用类名的前缀应该是他们属于的组件的名称（就像在C或Objective-C语言，创建“非正式”命名空间）。

3）应避免用全局选择器。因为在单个页面内，一个部件可以多次使用（一个例子是OpenERP的仪表板），选择范围应限制在一个给定的组件的范围。未过滤的选择，如$(selector) or document.querySelectorAll(selector) 通常会导致意外或不正确的行为。OpenERPWeb的部件有个提供 DOM 根标签的属性的( Widget.$el )和一个快捷直接选择节点的( Widget.$())。

4)更一般的是，永远不要假设你自己的组件拥有或能够控制任何超出自己的$el。

5)除非绝对是微不足道的，应该使用 QWeb 进行HTML模板和渲染。

6)所有交互式组件（屏幕上显示信息的组件、或拦截DOM事件的组件）都必须继承自部件，正确的执行、使用它的API，符合生命周期。

[● qweb](http://odoodevelop.lofter.com/tag/qweb)[● openerp](http://odoodevelop.lofter.com/tag/openerp)[● odoo](http://odoodevelop.lofter.com/tag/odoo)[● js](http://odoodevelop.lofter.com/tag/js)

[热度(1)全文链接](http://odoodevelop.lofter.com/post/1cd05c40_41e3faa)

[05](http://odoodevelop.lofter.com/post/1cd05c40_41e318f)

[12](http://odoodevelop.lofter.com/post/1cd05c40_41e318f)

[高德地图应用](http://odoodevelop.lofter.com/post/1cd05c40_41e318f)

    在openerp中的fleet模块，每一个车辆都有地图应用。默认采用的是谷歌地图，但是在应用得时候如果想换其他的地图该怎么做那？以下我们以国内市面上比较流行的高德地图为例，给大家分析一下如何换地图。

**一，openerp的地图设计思路**

    openerp的地图应用是运用qweb来实现的。在xml中声明一个div，然后在js中初始化地图放到声明的div里即可。在js中会把这一系列的初始化地图，加载等动作注册为一个openerp的widget。最后在相关的视图页面引用这个widget即可。所有，我们如果要运用新的地图，我们只需要修改地图相关的js和xml即可。

**二，高德地图实现**

    在进行修改之前，我们首先要把高德地图的javascript的api复制到一个静态文件之中。然后在进行如下代码的修改，修改完后在\_\_open

erp\_\_.py中导入gaodeapi.js，gaode.js，gaode.xml即可。

**gaode.xml：**

    <?xml version="1.0" encoding="UTF-8"?>

    <templates xml:space="preserve">

    <t t-name="WidgetGaodeCoordinates">

           <div id="container" class="gaode"></div>

    </t>

    </templates>

**gaode.js：**

    openerp.fleet = function(instance) {

    var \_t = instance.web.\_t,

        \_lt = instance.web.\_lt;

    var QWeb = instance.web.qweb;

    instance.fleet = {};

    instance.web.form.widgets.add('color', 'instance.oepetstore.FieldColor');

    instance.fleet.WidgetGaodeCoordinates = instance.web.form.FormWidget.extend({

        init: function() {

            this.\_super.apply(this, arguments);

        },

        start: function() {

            this.\_super();

            this.field\_manager.on("field\_changed:provider\_latitude", this, this.display\_map);

            this.field\_manager.on("field\_changed:provider\_longitude", this, this.display\_map);

            this.on("change:effective\_readonly", this, this.display\_map);

            this.display\_map();

        },

        display\_map: function() {

            var self = this;

            var provider\_latitude = this.field\_manager.get\_field\_value("provider\_latitude");

            var provider\_longitude = this.field\_manager.get\_field\_value("provider\_longitude");

            this.$el.html(QWeb.render("WidgetGaodeCoordinates", {

                "latitude": provider\_latitude || 0,

                "longitude": provider\_longitude || 0,

            }));

            var mapObj = new AMap.Map("container");

            mapObj.plugin(["AMap.ToolBar","AMap.OverView","AMap.Scale"],function(){

                //加载工具条

                tool=new AMap.ToolBar({

                  direction:false,//隐藏方向导航

                  ruler:false,//隐藏视野级别控制尺

                  autoPosition:false//禁止自动定位

                });

                mapObj.addControl(tool);

                //加载鹰眼

                view=new AMap.OverView();

                mapObj.addControl(view);

                //加载比例尺

                scale=new AMap.Scale();

                mapObj.addControl(scale);

              });

            var marker = new AMap.Marker({

                id:"m",

                position:new AMap.LngLat(provider\_latitude,provider\_longitude),

                offset: new AMap.Pixel(-8,-34),

                icon: "http://webapi.amap.com/static/images/marker\_sprite.png",

                level: 15

            });

            var point = new AMap.LngLat(provider\_latitude, provider\_longitude);

            mapObj.setCenter(point);

            mapObj.addOverlays(marker);

        },

    });

    instance.web.form.custom\_widgets.add('coordinates', 'instance.fleet.WidgetGaodeCoordinates');

}

[● odoo](http://odoodevelop.lofter.com/tag/odoo)[● openerp](http://odoodevelop.lofter.com/tag/openerp)[● qweb](http://odoodevelop.lofter.com/tag/qweb)[● js](http://odoodevelop.lofter.com/tag/js)

[热度(1)全文链接](http://odoodevelop.lofter.com/post/1cd05c40_41e318f)

[05](http://odoodevelop.lofter.com/post/1cd05c40_41df7fd)

[12](http://odoodevelop.lofter.com/post/1cd05c40_41df7fd)

[Hide "Activate the Developer Mode" Link](http://odoodevelop.lofter.com/post/1cd05c40_41df7fd)

Create xml file under \*\***static/src/base.xml**\*\* in your module and add this code:  
  
    <templates>  
        <t t-extend="UserMenu.about">  
            <t t-jquery="a.oe\_activate\_debug\_mode" t-operation="replace"/>  
        </t>  
    </templates>  
  
Add this file in \*\***\_\_openerp\_\_.py**\*\*:  
  
    'qweb' : [  
        "static/src/xml/your.xml",  
    ]

Now restart the server and update your database and refresh the page.

This code will remove \*\***Activate the Developer Mode**\*\* link inside \*\*About OpenERP\*\* option.

[● js](http://odoodevelop.lofter.com/tag/js)[● qweb](http://odoodevelop.lofter.com/tag/qweb)[● odoo](http://odoodevelop.lofter.com/tag/odoo)[● openerp](http://odoodevelop.lofter.com/tag/openerp)

[全文链接](http://odoodevelop.lofter.com/post/1cd05c40_41df7fd)

[04](http://odoodevelop.lofter.com/post/1cd05c40_415e1b6)

[12](http://odoodevelop.lofter.com/post/1cd05c40_415e1b6)

[record里面的id到底存在哪里](http://odoodevelop.lofter.com/post/1cd05c40_415e1b6)

    我们在做OE二次开发的时候不管是创建权限组，还是创建视图，都是要用到，record。而我们在用record的时候经常会给他起一个id来唯一的标识他，如下面所示：

 <record model="res.groups" id="group\_staff">

        <field name="name">Staff User</field>

        <field name="comment">the staff groups</field>

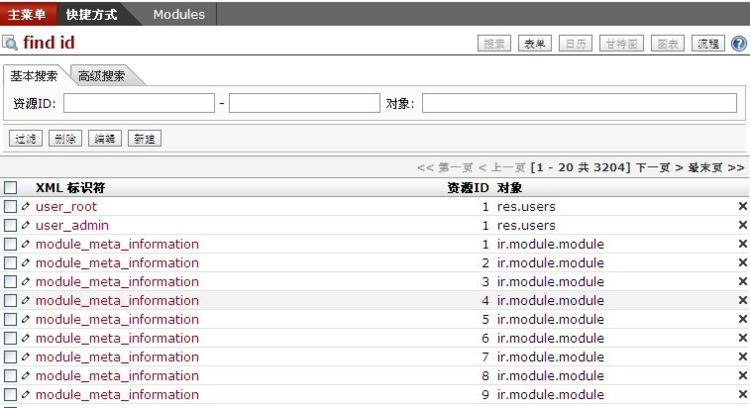
</record>

    但是我们在res.groups这个组里找了半天也没有找到这个id为group\_staff的字段，很是费解！其实这个id是存在ir\_model\_data这个表中的，如下是他的一条记录：

    我们可以看到这个name就是我们所创建的 id号，分析一下这条记录，来看看这个表的作用：

    name就是我们要的字符型的 id  
    model是这条记录所属的对象名  
    res\_id是这条记录在它的model所在的数据表里的database id，也就是那个唯一的主键

    那么去哪里得到当前记录的model和res\_id呢？  
    哈哈，地址栏  
  
    这么有用的表，每次都到数据库里去sql就太笨了吧，好在我们可以给 对象ir.model.data建个菜单，再自定义视图把model和res\_id设置为 总是可搜索。



[● 视图](http://odoodevelop.lofter.com/tag/%E8%A7%86%E5%9B%BE)[● record](http://odoodevelop.lofter.com/tag/record)[● openerp](http://odoodevelop.lofter.com/tag/openerp)[● odoo](http://odoodevelop.lofter.com/tag/odoo)

[热度(2)全文链接](http://odoodevelop.lofter.com/post/1cd05c40_415e1b6)

[03](http://odoodevelop.lofter.com/post/1cd05c40_4140f54)

[12](http://odoodevelop.lofter.com/post/1cd05c40_4140f54)

[网上的OpenERP-XML-RPC的Jar包应用](http://odoodevelop.lofter.com/post/1cd05c40_4140f54)

OpenERP的每一个模块里都有相应的增删改查的方法供我们来对数据库的数据进行操作，而有时候我们希望在远程调用某一个模块中的相应方法，比如我们移动终端设备上的调用。这个时候，我们可以运用XML-RPC进行方法的调用，下面对用XML-RPC对OpenERP进行数据操作的方法介绍：

**连接操作：**

public class Connect{

    public int Connect(String host,int port,String db,String username,String password){

        int flag = 0;

        XmlRpcClient xmlrpclogin = new XmlRpcClient();

        XmlRpcClientConfigImpl xmlrpcConfigLogin = new XmlRpcClientConfigImpl();

        xmlrpcConfigLogin.setEnabledForExceptions(true);

        try {

            xmlrpcConfigLogin.setServerURL(new URL("http", host, port,

                    "/xmlrpc/common"));

        } catch (MalformedURLException e1) {

            // TODO Auto-generated catch block

            e1.printStackTrace();

        }

        xmlrpclogin.setConfig(xmlrpcConfigLogin);

        try {

            Object[] params = new Object[] { db, username, password };

            Object id = xmlrpclogin.execute("login", params);

            if (id instanceof Integer)

                flag = (Integer) id;

        } catch (Exception e) {

            e.printStackTrace();

        }

        return flag;

    }

}

**查找操作：**

public class Search{

    public Object Search(String host,int port,String db,String username,String password,String carCode,int id){

        XmlRpcClient xmlrpc = new XmlRpcClient();

        XmlRpcClientConfigImpl xmlrpcConfgDb = new XmlRpcClientConfigImpl();

        xmlrpcConfgDb.setEnabledForExceptions(true);

        try {

            xmlrpcConfgDb.setServerURL(new URL("http", host, port,

                    "/xmlrpc/object"));

        } catch (MalformedURLException e1) {

            // TODO Auto-generated catch block

            e1.printStackTrace();

        }

        xmlrpc.setConfig(xmlrpcConfgDb);

        Object[] l = { "license\_plate","=",carCode };

        Object[] filters = { l };

        // filters.add(l);

        Object[] params = new Object[] { db, id, password, "fleet.vehicle",

                "search", filters };

        Object[] o = null;

        try {

            o = (Object[]) xmlrpc.execute("execute", params);

        } catch (Exception e) {

            e.printStackTrace();

        }

        xmlrpcConfgDb = null;

        return o[0];

    }

}

**修改操作：**

public void writeObject(int uid,String model,HashMap<String, Object> values,ArrayList ids) throws MalformedURLException {

            XmlRpcClient xmlrpc = new XmlRpcClient();

            XmlRpcClientConfigImpl xmlrpcConfgDb = new XmlRpcClientConfigImpl();

            xmlrpcConfgDb.setEnabledForExceptions(true);

            xmlrpcConfgDb.setServerURL(new URL(

                    "http://"+appGlobal.getHostname()+":"+appGlobal.getPort()+"/xmlrpc/object"));

            xmlrpc.setConfig(xmlrpcConfgDb);

            String dbname = appGlobal.getDbname();

            String pwd = appGlobal.getPassword();

            Object[] params = new Object[] { dbname, uid, pwd, model,

                    "write", ids,values };

            try {

                xmlrpc.execute("execute", params);

            } catch (Exception e) {

                e.printStackTrace();

            }

        }

}

**总结：**

在进行调用的时候，通常会遇到类型转化的错误，这是由于在调用方法的时候传输参数出现错误的情况，比如我们想对openerp里的fleet.vehicle数据表某个字段进行修改。那么这个时候我们需要知道要修改的行号，我们可能会传个int类型过去，但是在openerp端，write()方法中的参数里ids的类型却为List类型，所以这个时候就会报数据类型转化的错误。所以在进行调用的时候一定要弄清楚服务器端需要的参数类型是什么样的。

[● openerp](http://odoodevelop.lofter.com/tag/openerp)[● odoo](http://odoodevelop.lofter.com/tag/odoo)[● xml-rpc](http://odoodevelop.lofter.com/tag/xml-rpc)

[全文链接](http://odoodevelop.lofter.com/post/1cd05c40_4140f54)

[03](http://odoodevelop.lofter.com/post/1cd05c40_4142570)

[12](http://odoodevelop.lofter.com/post/1cd05c40_4142570)

[OpenERP中的XML-RPC客户端](http://odoodevelop.lofter.com/post/1cd05c40_4142570)

**一，应用准备**

    在实际的业务中我们往往会开发一个客户端，来和OpenERP的服务端进行交互。比较典型的例子就是Android客户端的实现。在以上了解了OpenERP中XML-RPC服务端的实现方式以后，我们要做的就是弄清楚客户端开发需要哪些必备的包以及如何调用方法。我们知道Android是用Java来写的，那么很显然我们就需要基于Java的服务端RPC方法调用的包。值得庆幸的是，网上已经有人为我们做好了这一切。我们可以在网上下载到名为openerp-java-api-1.3.0.jar的包。把它导进Android的开发平台里，调用相关的方法即可。具体的方法调用都写在了openerp-java-api-1.3.0-javadocs.zip这个帮助文档里。

    不过在我们的实际应用中应用这个包总是会出现间歇性的连接失败，或者出现长时间的加载。所以，结合xmlrpc的java包与OpenERP服务端的RPC方法，我也打了一个类似的jar包。以下会以这个jar包为基础给大家讲解如何调用OpenERP服务端方法。

**二，调用详解**

**1，介绍jar包的结构**

    打的jar包名为OpenERP-java-jar.jar，它应用了xml-rpc的java实现的几个包，分别是ws-commons-util-1.0.2.jar，xmlrpc-common-3.1.3.jar，xmlrpc-server-3.1.3.jar，xmlrpc-client-3.1.3.jar。

    在OpenERP-java-jar.jar中主要了解OpenERPXmlRpcClient.java和OpenERPXmlRpcService.java这两个文件，第一个文件提供了创建客户端的方法createClient（args），我们调用的时候只需要传入URL的各个部分即可；第二个部分集中了如何调用服务器端web\_services里的方法。下面将通过一个例子来为你解释如何调用。

**2，例子详解方法调用**

  （1），将我们打的jar包导入新建的项目之中

  （2），新建jarTest.java文件，写入如下的代码：

   package com.test;

   import java.util.ArrayList;

   import com.openerp.jar.OpenERPXmlRpcService;

   public class jarTest {

        public static void main(String[] args){

              OpenERPXmlRpcService ss = new OpenERPXmlRpcService("http","115.28.247.88",8072,"xxxxx","admin","xxxxx");

              ArrayList finalResults = ss.getDatabaseList();

              System.out.println(finalResults.size());

        }

   }

   这里我们声明了一个OpenERPXmlRpcService对象，在里面我们传入了它需要的各个参数protocol，host,port,db,uname,pwd

。然后在对象创建以后我们调用了它其中的一个getDatabaseList()的方法用来返回OpenERP中所有数据库的信息。最后我们打印了这个list的大小，得出了当前OpenERP服务端总共创建了多少个数据库。这就是一个简单的调用过程，其他的调用都和此差不多，方法都在OpenERPXmlRpcService.java这个文件之中。可以结合服务器端的方法来看，会更清楚他们之间的通信方式。

   当然，如果你想用网上的jar包也可以。不过他里面的方法需要反编译才能看得到。网上java的反编译软件有的是，可以看看人家是如何实现的。

[● openerp](http://odoodevelop.lofter.com/tag/openerp)[● odoo](http://odoodevelop.lofter.com/tag/odoo)[● xml-rpc](http://odoodevelop.lofter.com/tag/xml-rpc)

[全文链接](http://odoodevelop.lofter.com/post/1cd05c40_4142570)

[03](http://odoodevelop.lofter.com/post/1cd05c40_41421df)

[12](http://odoodevelop.lofter.com/post/1cd05c40_41421df)

[OpenERP中的XML-RPC服务端](http://odoodevelop.lofter.com/post/1cd05c40_41421df)

**一，简介**

    OpenERP运用了WSGI的Web服务器协议，另外值得称道的是在OpenERP中支持了XML-RPC协议。这样我们在客户端就可以运用不同的编程语言和服务器端进行交互。因为RPC的实现有各种语言版本，从python，java到c#，几乎我们平时用到的主流语言都有其实现。但是想要比较准确无误的在实现客户端和服务器端之间的通信，首先我们要做的就是了解服务器端RPC的实现代码中都列举了哪些的方法，如何定位到这些方法。下面来介绍一下OpenERP在服务端的XML-RPC的实现方式。

**二，实现方式**

    在OpenERP中了解RPC主要看service目录下的wsgi\_server.py和web\_services.py，其中wsgi\_server.py主要对传进来的URL进行匹配，然后分发到web\_services.py的相应方法之中，执行方法，返回结果。下面来看一下具体的代码实现：

**1，请求进入WSGI入口点**

    在wsgi\_server.py的385行定义的application\_unproxied方法即是请求进行解析处理的入口点，我们可以看到以下的代码：

    wsgi\_handlers = [wsgi\_xmlrpc\_1, wsgi\_xmlrpc, wsgi\_xmlrpc\_legacy, wsgi\_webdav]

    wsgi\_handlers += module\_handlers

    for handler in wsgi\_handlers:

        result = handler(environ, start\_response)

        if result is None:

            continue

        return result

    这段代码对当前的请求在wsgi\_xmlrpc\_1, wsgi\_xmlrpc, wsgi\_xmlrpc\_legacy, wsgi\_webdav四种handlers模式下进行了分析处理，如果有返回结果，就会返回result。

**2，service以及method的转发**

    以上的四种handlers着重看wsgi\_xmlrpc\_1这种，因为它是主要的WSGI处理方法，其中就包含了如何把请求URL转发到某个services下的某个method中。

    我们在wsgi\_xmlrpc\_1方法中可以看到service主要有三个分类，分别是db，common，object。然后通过params, method = xmlrpclib.loads(data)这一句获得的method分发到相应的方法中。

**3，执行相应的方法，返回结果**

    service的具体方法在web\_services.py这个文件当中，其中很显眼的划分了db,common,object三个类，而每一个类里就定义了相应的方法。比如，我们的Android客户端要和OpenERP服务器进行通信，那么首先就需要执行登录的方法。这个时候，我们传入的URL就会是这样的：http://localhost:8069/common/login，就是找服务common下的login方法执行登录。具体的方法操作会在接下来一节中为你详说。

[● openerp](http://odoodevelop.lofter.com/tag/openerp)[● odoo](http://odoodevelop.lofter.com/tag/odoo)[● xml-rpc](http://odoodevelop.lofter.com/tag/xml-rpc)

[热度(1)全文链接](http://odoodevelop.lofter.com/post/1cd05c40_41421df)

[03](http://odoodevelop.lofter.com/post/1cd05c40_413f98d)

[12](http://odoodevelop.lofter.com/post/1cd05c40_413f98d)

[RPC的客户端以及服务端](http://odoodevelop.lofter.com/post/1cd05c40_413f98d)

**一，通信机制**

    一个rpc系统必然包括两个部分，rpc client（用来向rpc server调用方法，并接收方法的返回数据），rpc server（用于响应rpc client的请求，执行方法，并回送方法执行结果），他们之间的通信过程也就显而易见：

    rpc client的工作原理:rpc client根据URL找到rpc server -> 构造命令包，调用rpc server上的某个服务的某个方法 -> 接收到rpc server的返回,解析响应包，拿出调用的返回结果。

    rpc server的工作原理:启动一个web server(在使用内置的web server的情况下) -> 注册每个能提供的服务，每个服务对应一个Handler类 ->进入服务监听状态。

    这两个部分之间的通信我们也称之为“分布式计算”，提高了各个分立机器的互操作性。

**二，实现例子**

    以上介绍了XML-RPC客户端以及服务端的通信机制，理论需要建立在代码之上方能目睹其真实的效果。下面就以python为例，创建一个服务端和一个客户端，并进行通信。

    服务端：

    import calendar, SimpleXMLRPCServer

    class Calendar:

          def getMonth(self, year, month):

              return calendar.month(year, month)

    calendar\_object = Calendar()

    server = SimpleXMLRPCServer.SimpleXMLRPCServer(("localhost", 8888))

    server.register\_instance(calendar\_object)

    server.serve\_forever()

    客户端：

    import xmlrpclib

    server = xmlrpclib.ServerProxy("[http://localhost:8888")](http://localhost:8888%22)/)

    month = server.getMonth( 2014, 4 )

    print month

    python自带了SimpleXMLRPCServer和xmlrpclib用来创建RPC的服务端以及客户端，显然简单的多了。通过上面的例子，你应该能够感受得到RPC的协议机制了吧。