OFBiz Tutorial - A Beginners Development Guide

OFBiz 教程 – 初学者的开发指南

http://docs.ofbiz.org/display/OFBIZ/OFBiz+Tutorial+-+A+Beginners+Development+Guide

Translated by hongs， contact： billhongs@gmail.com

Creating Practice Application (Hello World...)

Written by: Pranay Pandey with feedback and contributions from Chirag Manocha, Ravindra Mandre, Rob Schapper

Special thanks to Anil Patel and Mridul Pathak for inspiring me to write this tutorial.

这教程面向 OFBiz 初学者。帮助学习 OFBiz 应用开发流程基础。创建这个教程背后的目的是让开发者通过最佳实践熟悉代码习惯、控制流或更多。这个练习充当 OFBiz 的”Helloworld”，和由 Kernighan and Richie 引入的 c 语言编程的第一个 Helloworld 一样。

重要！

这教程打算使用最新的 SVN 版本。不再适用于 4 版本。

与这个应用开发的同时，你可以看 OFBiz 的视频，它们在 http://docs.ofbiz.org/display/OFBTECH/Framework+Introduction+Videos+and+Diagrams,

Part - 1

注 1: 有更多的疑问，你可以参考 Example 组件。你准能在 OFBiz 的最新代码的 example 组件中找到你想要的代码。任意时候想参考例子代码，作为这个应用的开发，这些将给你未来的开发中带来帮助。每一个新功能会第一时间加入到 example 组件中作为参考。

注 2：在开始开发这应用前，你一定要阅读下面内容：OFBiz Contributors Best Practices（http://docs.ofbiz.org/display/OFBADMIN/OFBiz+Contributors+Best+Practices）， Coding Conventions（http://docs.ofbiz.org/display/OFBADMIN/Coding+Conventions）and Best Practices Guide（http://docs.ofbiz.org/display/OFBADMIN/Best+Practices+Guide）。.

注 3：不要从其它组件中拷贝文件，因为修订的版本号也会拷贝。如果需要总是创建一个新文件，然后拷贝这文件的内容。也同时可以注意到无用的代码。
注 4：搜索任意文档最好的地址是 http://docs.ofbiz.org/display/OFBADMIN/OFBiz+Documentation+Index。
注 5：从右边开始阅读控制台的日志是一个习惯，可以使得排除问题变得容易些，更好地理解系统。

创建第一个基本应用，命名为"practice"

第 1 步：在 hot-deploy 下创建子目录命名为“practice“。这个目录名要匹配我们要创建的组件名。
注：记住所有的定制化开发都仅仅在这完成。
第 2 步：在 hot-deploy/practice 路径下创建 ofbiz-component.xml 文件，填入下面的内容。（作为参考你可以检查这个文件在 OFBiz 任意其它组件中）

```
<?xml version="1.0" encoding="UTF-8"?>
<ofbiz-component name="practice"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://ofbiz.apache.org/dtds/ofbiz-component.xsd">
      <resource-loader         name="main" type="component"/>
    <webapp name="practice"
       title="Practice"
       server="default-server"
       base-permission="OFBTOOLS"
       location="webapp/practice"
       mount-point="/practice"
       app-bar-display="false"/>
</ofbiz-component>
```

ofbiz-component.xml 说明:

1. 这个 **ofbiz-component.xml 文件是负责让 OFBiz 知道资源的位置，同时让你加入到 classpath 中**.

2. 这个'resource-loader name' 可以是任意字符串。这儿我们设为"main". 这个 'type' 告诉 OFBiz 我们将开始装载一个组件。

```
<resource-loader name="main" type="component"/>
```

3. 在**<webapp>标签中,有不同的属性和它们的目的如下：**

```
<webapp name="practice"
        title="Practice"
        server="default-server"
        base-permission="OFBTOOLS"
        location="webapp/practice"
        mount-point="/practice"
        app-bar-display="false"/>
```

a) name :- 定义我们 web 应用的名字。
b) title :- 这个将是应用的标识，会显示在顶端导航栏上。
c) server :- 这个让 OFBiz 知道使用哪个 server
d) base-permission :- 这行要求用户要有 OFBTOOLS 权限才能使用这个应用。因为'admin'用户有这个权限，所以我们不必创建其它新用户。
e) location :- 在这个服务器上缺省基准路径的位置。
f) mount-point :- 这是访问资源的 URL，应该是 localhost:8080/practice
g) app-bar-display :- 这个是让 OFBiz 知道是否显示在主应用导航条上，这个是公用 OFBiz 修饰的一部分。

创建 web 应用

Step - 1 : 在 practice 组件中创建一个"webapp"目录(hot-deploy/practice/webapp).，这个目录包含所有这组件相关的 webapp 目录。
Step - 2 : 在 webapp 目录下创建一个子目录命名为"practice",这个就是我们要开发的 webapp 名称(hot-deploy/practice/webapp/practice). 一个组件可以附加多个 web 应用。比如在**"marketing"组件中有两个** webapps **"marketing"** and **"sfa"**。我们创建的 webapp 将按照 J2EE 的 Web 应用标准。
Step - 3 : 在你 webapp 下创建 WEB-INF 目录(hot-deploy/practice/webapp/practice/WEB-INF)。一个 OFBiz 的 web 应用要有两个配置文件：**controller.xml** 和 **web.xml**。这 controller.xml 告诉 OFBiz 从访问者来的不同请求做不同的事：做什么动作和渲染什么页面。web.xml 告诉 OFBiz 什么资源 (database and business logic access) 对这个 web 应用是有效的和如何处理 web 相关的事，比如 welcome pages, redirects, and error pages。.
Step - 4 创建一个命名为"web.xml"(web.xml 遵守 j2ee web 应用规范). 这个文件的内容可以从存在的任意组件中拷贝，比如 /framework/example 组件. 需要改变的重要值是<display-name>、**localDispatcherName** 和 **mainDecoratorLocation**.

```
<context-param>
    <param-name>localDispatcherName</param-name>
    <param-value>practice</param-value>
    <description>A unique name used to identify/recognize the local dispatcher for the Service Engine</description>
</context-param>
<context-param>
```

```
    <param-name>mainDecoratorLocation</param-name>
    <param-value>component://practice/widget/CommonScreens.xml</param-value>
    <description>The location of the main-decorator screen to use for this webapp; referred to as a context variable in screen def XML files.</description>
</context-param>
```

现在设置这个值: "component://practice/widget/CommonScreens.xml" 作为这个位置，你一会就会知道原因。这个位置是用来指出在 screens 中的主修饰器位置的。象如下：

${parameters.mainDecoratorLocation}，通过改变它增加了代码的独立性，当我们需要改变主修饰器的位置。那时我们仅仅需要改为这个位置，将会应用到使用它的所有 screen。在 screen 中这样做的好处是可以从其它组件中重用已存在的 screen，但是它用你的修饰器来修饰那个 screen，仅仅是在相同模式用在所有地方和在所有组件中来显示 mainDecoratorLocation.在这个应用开发的不远将来，当你在你的 screen 中增加修饰器的时候会集中到这点上。

Step - 5 创建一个命名"controller.xml" (被 ofbiz webapp 控制器使用)的文件。 这个文件开始是小而简单，但随后我们增加功能而快速增长。现在插入下列代码:

```
<?xml version="1.0" encoding="UTF-8"?>
<site-conf xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="http://ofbiz.apache.org/dtds/site-conf.xsd">
        <include location="component://common/webcommon/WEB-INF/common-controller.xml"/>
        <description>Practice Component Site Configuration File</description>
        <owner>Copyright 2001-2009 The Apache Software Foundation</owner>
<handler name="screen" type="view" class="org.ofbiz.widget.screen.ScreenWidgetViewHandler"/>        <!-- Request Mappings -->
        <request-map uri="main">
            <security https="false" auth="false"/>
            <response name="success" type="view" value="main"/>
        </request-map>
        <!-- end of request mappings -->
        <!-- View Mappings -->
        <view-map name="main" type="screen" page="component://practice/widget/PracticeScreens.xml#main"/>
        <!-- end of view mappings -->
</site-conf>
```

Step - 6 : 移到上一级，创建一个新的目录，命名为'error'(hot-deploy/practice/webapp/practice/error).
Step - 6.a : 在"error"目录中创建一个文件. 这个文件的内容可以取自任意存在的组件中，比如 example 组件。你显示错误信息页面的位置将被指定在 controller.xml 文件的开始位置，比如 <errorpage>/error/error.jsp</errorpage> .你需要创建或拷贝一个/webapp/practice/error/error.jsp 向用户显示错误信息。
Step - 7 : 在你的组件目录 practice 中创建一个"widget"(hot-deploy/practice/widget). 这个目录就包含 forms, menus, and screens，用来处理用户界面的。
Step - 8 : 在"widget"中创建文件"PracticeScreens.xml".这个文件的内容可以取自任意组件中，例如 example 组件。自从现在开始，你将能开始创建 screens 视图，所以在这个地方一个重要的阅读是“Best Practices Guide（http://docs.ofbiz.org/display/OFBADMIN/Best+Practices+Guide）”。 在这个页上链接到下面：

- HTML and CSS Best Practices
- Managing Your Source Differences
- Methodology Recommendations
- User Interface Layout Best Practices
  所有这些阅读是真的有帮助。
  最有可能开始的 screen 将会是如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<screens xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://ofbiz.apache.org/dtds/widget-screen.xsd">
    <screen name="main">
        <section>
            <widgets>
                <label text="This is first practice"/>
            </widgets>
        </section>
    </screen>
</screens>
```

Step - 9 : 现在我们这里有基本的元素，让我们再检查基本流程，不管你的组件变得如何庞大和复杂。开始，通过浏览器来发动一个请求来看指定的资源。举例说明这个请求：*"localhost:8080/practice/control/main"*

1. 当 OFBiz 看到这个请求，首先检查/practice 部分。这是因为在我们的 ofbiz-component.xml 文件，我们指出我们的 web 应用的装载点是/practice。现在知道我们的 practice 组件将处理余下的请求。
2. OFBiz 将然后看我们的 controller.xml 文件。在我们的 controller.xml 文件有 request-maps 和 view-maps。如果它发现一个名字为'main'的 request-map，它将使用相关联的 view-map，如下所述。这个 request-map 也能指定一个 view 或我们将会看到的 event 或 service. 如果指定一个 view 将在 the controller.xml 文件中进一步查找，看是否有在 request-map 指定值对应的 view-map。
3. 现在我们继续保持简单，假定所有 views 到一个 type=screen. 在这个情况下，页面标签指定一个路径到 screen 定义文件而且一个 screen 名称显示在"#"符号后。

Step - 10 : 现在该运行你的第一个练习应用。

通过命令行输入下面: **java -Xmx256M -jar ofbiz.jar** (the -Xmx256M 命令仅保证程序有足够的内存)。然后在浏览器上点击这个地址 http://localhost:8080/practice/control/main。浏览器应该看到输出屏幕显示"This is first practice"**:**

Step - 11 : 在 webapp 目录"practice"创建一个 index.jsp 文件 (文件的内容能拷贝自"example"组件).这个文件负责重定向响应至 control/main,如果你给一个象如下的 URL http://localhost:8080/practice/。如果你给一个象如下的 URL：http://localhost:8080/practice/unknown/requestit 将会重定向到 web.xml 文件中指定的 redirectPath。.既然那样, ContextFilter 将过滤出这个请求和使用这个重定向路径来重定向这个请求。

Part - 2

做一些高级的

Step - 1 : 现在是该在这个应用中为 screens 创建 decorator。在"widget"目录中创建一个 CommonScreens.xml。这个文件包括公用 screens，用来贯穿整个应用。一个公用 screen 可以有一个 header 和 footer 被包含以便任意其它 screens 能用它作修饰器也同样有这些项。这些你可以参考"example"组件中的 CommonScreens.xml 文件。

CommonScreens.xml 文件中代码将是：

```
<screen name="CommonPracticeDecorator">
    <section>
        <widgets>
            <decorator-section-include name="body"/>
        </widgets>
    </section>
</screen>
```

参照"example"组件中的 CommonScreens.xml 文件 main-decorator 的用法。这一刻两个重要的阅读点是：Understanding the OFBiz Widget Toolkit(http://docs.ofbiz.org/display/OFBIZ/Understanding+the+OFBiz+Widget+Toolkit) 和 FAQ(http://docs.ofbiz.org/display/OFBIZ/FAQ+-+Tips+-+Tricks+-+Cookbook+-+HowTo)的修饰器部分。

Step - 2 : 为这个应用创建一个菜单。为此在你的组件的"widget"目录中创建一个 PracticeMenus.xml 文件。这个可以参考"example"组件的 ExampleMenus.xml 文件。

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

<menus xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://ofbiz.apache.org/dtds/widget-menu.xsd">

    <menu name="MainAppBar" title="PracticeApplication" extends="CommonAppBarMenu"
extends-resource="component://common/widget/CommonMenus.xml">

        <menu-item name="main" title="Main"><link target="main"/></menu-item>

    </menu>

</menus>
```

如下在你的 CommonPracticeDecorator 包含菜单文件:

```
<screen name="CommonPracticeDecorator">

        <section>

            <widgets>

                <include-menu
location="component://practice/widget/PracticeMenus.xml" name="PracticeAppBar"/>

                <decorator-section-include name="body"/>

            </widgets>

        </section>

    </screen>
```

Step - 3: 在 WEB-INF 目录中创建子目录"actions"。在这个目录中我们将创建脚本文件。脚本文件是准备数据的。这些文件将是 groovy 文件。以前我们是用 bsh(beanshell) 文件。这是用来在运行中从数据库中取数据给用户界面的脚本。参考： Tips & Tricks while working with Groovy(http://docs.ofbiz.org/pages/viewpage.action?pageId=4472) 和 http://groovy.codehaus.org/. 工作在 groovy 中总是要注意导入类和包。仅导入在你文件中使用的。要从"Debug"类使用方法来放置日志消息，仅从它自己开始做起。因此在 actions 目录中创建一个 Person.groovy，用来从实体"Person"取出所有记录。这时实际上做这些只要很少代码 (一行 a single line) ，如下

```
context.persons = delegator.findList("Person", null, null, null, null, false);
```

上面的语句将从 Person 实体获取所有记录并通过名字 persons 放在 context。这个列表通过名称 person 在 ftl 文件中迭代显示记录。这方面要点是在：Which variables are available in screen context?(http://docs.ofbiz.org/pages/viewpage.action?pageId=4459&focusedCommentId=5533#comment-5533)

Step - 4 : 现在在"practice"的 web 应用中创建一个名称为"Person.ftl"的 ftl 文件，用来显示从 groovy 文件中取来的数据。参考：http://freemarker.sourceforge.net/docs/，这时你仅需迭代存在 context 中的 persons 列表。仅需要如下代码：

```
<#if persons?has_content>
```

```
  <h2>Some of the people who visited our site are:</h2>
  <br>
  <ul>
    <#list persons as person>
      <li>${person.firstName?if_exists} ${person.lastName?if_exists}<li>
    </#list>
  </ul>
</#if>
```

Step - 5 : 现在在 PracticeScreens.xml 文件中创建一个"person"的 screen，也在 PracticeMenus.xml 文件中创建一个新的菜单项。

PracticeScreens.xml 的新 screen 输入将是：

```
    <screen name="person">
        <section>
            <actions>
                <script
location="component://practice/webapp/practice/WEB-INF/actions/Person.groovy"/>
            </actions>
            <widgets>
                <decorator-screen name="CommonPracticeDecorator"
location="${parameters.mainDecoratorLocation}">
                    <decorator-section name="body">
                        <platform-specific>
                            <html>
                                <html-template
location="component://practice/webapp/practice/Person.ftl"/>
                            </html>
                        </platform-specific>
                    </decorator-section>
                </decorator-screen>
            </widgets>
        </section>
    </screen>
```

Step - 6 : 改变 controller.xml 文件指向我们刚弄的新 screen。输入 http://localhost:8080/practice/control/person . 来运行应用观察结果。

小提示

如果输出屏幕上没有包含下面的菜单，你最有可能在 controller.xml 中需要改变 auth="true"至 auth="false"，从而让菜单显示出来。

输出 Screen :

在 screen 上创建一个 form 来显示 Person 实体的内容。(使用 Form 部件)

Step - 1 : 现在加入一个或更多菜单项通过名称"PersonForm"至你的 PracticeMenus.xml 文件中

Step - 2: 在 widget 中创建一个 PracticeForms.xml 的文件，和创建一个列表 form 用来显示 person 实体的记录。(参考 ExampleScreens.xml 和 ExampleForms.xml 文件).

```
<form name="ListPersons" type="list" list-name="persons" list-entry-name="person"
target="updatePracticePerson" paginate-target="personForm">
        <auto-fields-service service-name="updatePracticePerson"
default-field-type="edit" map-name="person"/>
        <field name="partyId"><hidden/></field>
        <field name="submitButton" title="Update" widget-style="smallSubmit"><submit
button-type="button"/></field>
        <field name="deletePracticePerson" title="Delete Person"
widget-style="buttontext">
        <hyperlink target="deletePracticePerson?partyId=${person.partyId}"
description="Delete"/>
      </field>
</form>
```

Step - 3 : 创建一个名称为 personForm 的新 screen，然后包含这个列表 form。

```
<screen name="PersonForm">
        <section>
            <actions>
                <set field="headerItem" value="personForm"/>
                <set field="titleProperty" value="PageTitlePracticePersonForm"/>
                <entity-condition entity-name="Person" list="persons"/>
            </actions>
            <widgets>
                <decorator-screen name="CommonPracticeDecorator"
location="${parameters.mainDecoratorLocation}">
                    <decorator-section name="body">
                        <label text="Person List" style="h2"/>
                        <include-form name="ListPersons"
location="component://practice/widget/PracticeForms.xml"></include-form>
                    </decorator-section>
                </decorator-screen>
            </widgets>
        </section>
</screen>
```

Step - 4 : 在 controller.xml 中做必要的工作来显示这个 screen。现在再运行这个应用，观察不同之处。迄今你已经影响 controller requests mappings, Screen widget, form widget, Decorator, Menus, groovy, ftl.

创建主修饰器来修饰这个应用：

Step - 1 : 在 CommonScreens.xml 文件中创建 screen，名称为"main-decorator"。(参考 Example 组件中的 CommonScreens.xml)

```
<screen name="main-decorator">
        <section>
            <actions>
                <property-map resource="CommonUiLabels" map-name="uiLabelMap"
global="true"/>
                <property-map resource="PracticeUiLabels" map-name="uiLabelMap"
global="true"/>
```

```
                <set field="layoutSettings.companyName"
from-field="uiLabelMap.PracticeCompanyName" global="true"/>
                <set field="activeApp" value="practice" global="true"/>
                <set field="applicationMenuName" value="PracticeAppBar"
global="true"/>
                <set field="applicationMenuLocation"
value="component://practice/widget/PracticeMenus.xml" global="true"/>
            </actions>
            <widgets>
                <include-screen name="GlobalDecorator"
location="component://common/widget/CommonScreens.xml"/>
            </widgets>
        </section>
</screen>
```

Step - 2 : 现在在已有的 CommonPracticeDecorator screen 包含这个修饰器。运行然后看有什么不同之处。

现在该在应用条(app bar)中显示 practice 应用

Step - 1 : 要达到这个效果只要在 ofbiz-component.xml 文件中设置 app-bar-display="true"。重启服务器再运行你就能在 app bar 看到 practice 应用。

创建用户界面标签 Create UI Labels:

*Step - 1 :在你的组件（比如："practice".）目录下创建一个名称为"config"的目录
Note: 记得在 ofbiz-component.xml 文件中为 config 目录配置一个条目，如下：

```
<classpath type="dir" location="config"/>
```

这意味着你通过访问配置文件把 config 目录放在类路径中。
Step - 2: 创建一个 PracticeUiLabels.xml 文件，为这个 practice 应用创建用户界面标签(参考 ExampleUiLabels.xml). 记往一件事就是你啥时在 UiLabels 中改变了，你必须重启服务器才能生效。首先只创建 2 个 ui labels 如下：

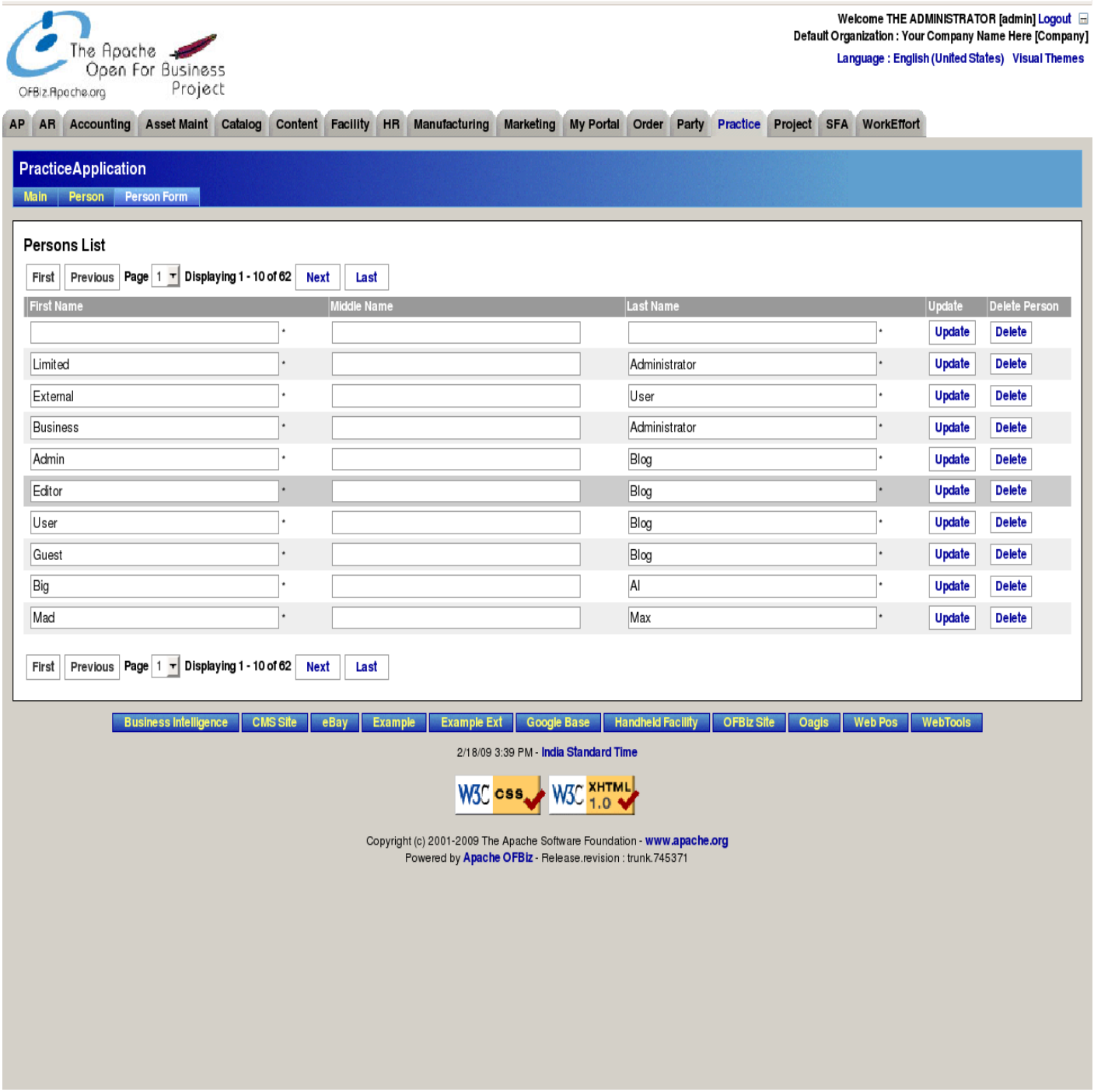
```
<property key="PracticeApplication">
    <value xml:lang="en">This is first        practice</value>
</property>
<property key="PracticeCompanyName">
```

```
    <value xml:lang="en">OFBiz: Practice</value>
</property>
```

Step - 3:包含资源在你早先创建好的修饰器 scrren 中，然后用一两个 ui labels。

Step - 4 : 在适当的地方使用这 2 个用户界面标签。

注:总是在 ofbiz 开始搜索任意存在 Ui label，如果你没找到然后仅创建一个新的输出屏幕(Output Screen):



现在该是通过检查验证（用户登录）来使这个练习应用安全:

Step - 1 :从 ExampleMenus.xml 文件中参考在你的菜单中加上 login and logout 选项。这些选项也会在 "component://common/webcommon/WEB-INF/common-controller.xml", 这个要包含在我们的 controller.xml，或者你自己加入这些项：

```
<!- Request Mappings ->

<!-- Security Mappings -->

 <request-map uri="checkLogin" edit="false">

    <description>Verify a user is logged in.</description>

        <security https="true" auth="false"/>

        <event type="java" path="org.ofbiz.webapp.control.LoginWorker"
invoke="checkLogin" />

        <response name="success" type="view" value="main"/>

        <response name="error" type="view" value="login"/>

    </request-map>

    <request-map uri="login">

        <security https="true" auth="false"/>

        <event type="java" path="org.ofbiz.webapp.control.LoginWorker"
invoke="login"/>

        <response name="success" type="view" value="main"/>

        <response name="error" type="view" value="login"/>

    </request-map>

    <request-map uri="logout">

        <security https="true" auth="true"/>

        <event type="java" path="org.ofbiz.webapp.control.LoginWorker"
invoke="logout"/>

        <response name="success" type="request" value="checkLogin"/>

        <response name="error" type="view" value="main"/>

    </request-map>
```

这些请求只有当你没有包含任何带有这些请求的其它组件时，加入你的控制文件中。所以如果你已经包含 common-controller.xml 文件，你自己不必在控制器中显式做这些的了，相同的 common-controller.xml 文件中的 view 也能被使用。我们也可以有自己的：

```
<view-map name="login" type="screen"
page="component://common/widget/CommonScreens.xml#login"/>
```

Step - 2 : 在 controller.xml 文件中 requests 修改使 auth="true"，意思是这些请求需要验证。这是你最开始安全级别，你已经实现的请求应该象这样：

```
<request-map uri="main">
    <security https="true" auth="true"/>
    <response name="success" type="view" value="main"/>
    <response name="error" type="view" value="main"/>
</request-map>
```

现在你运行应用，观察不同：你能用用户名 admin 和密码 ofbiz 登录。我们应该理解在 ofbiz-component.xml 文件中在基本权限中给的权限"OFBTOOLS"。. 要理解这些请仔细阅读下面然后按下做：
确认这个用户'admin'有'OFBTOOLS'权限。
Step : 1 – 登录 partymanager 来确认这个用户 admin 是否有要求的权限
https://127.0.0.1:8443/partymgr/control/main
Step : 2 – 一旦你登录后，在用户登录字段输入'admin'，然后点击查找会员(Lookup Party)按钮。
Step : 3 –这个执行通配符*搜索你将会看到多个用户返回，点详细按钮看 admin 用户。
注 ： 重要点是个人'Mr. THE PRIVILEGED ADMINISTRATOR'有一个会员标识 admin 在下面用户名表单中有多个登录帐号。
Step : 4 –我们对 admin 用户登录感兴趣，所以点击安全组'Security Groups'按钮，确认这个'admin'是'FULLADMIN'组的成员。用户所属的所有组显示在下面的列表表单中，在 FULLADMIN 上钻取(Drill down on the FULLADMIN).
Step : 5 – 点击权限 Permissions 标签页 tab. 这里显示 FULLADMIN 安全组的所有权限。在这些权限中浏览直到找到 OFBTOOLS 权限。'OFBTOOLS_VIEW 权限访问常用的 OFBiz 管理应用。' 这个证明 userlogin 'admin' 有权限'OFBTOOLS'
Step : 6 – 花一些时间来回顾下与用户和权限相关的实体模型。箭头表示关系的许多方面。一个十分重要的阅读在: OFBiz Security(http://docs.ofbiz.org/display/OFBTECH/OFBiz+security)

Part - 3

写 CRUD 操作:

创建、更新和删除操作一个实体将由 minilang 写成的服务实现。
开始的方法是我们写自己的服务来完成这些操作，这样来更好理解。接下来我们将通过调用已经实现的服务来完成。
要做这些，我们将采用会员模型的实体：
--Party
--Person
一个人是一会员，所以创建一个人先得创建一类型是 partyTypeId="PERSON"的会员。
所以有两种方式做这些：
1.创建一个服务来进行带有类型 Person 的会员创建工作。
2. 在服务中先创建一个会员，再创建个人。

写服务要按照下列步骤执行：

Step - 1:在组件目录"practice"创建"servicedef"目录。这个目录将包含所有的服务定义文件，比如：services.xml, secas.xml.
注:如果一个服务是用 java 编写的就放在"src" 目录，如果是用 minilang 编写的服务则放在 script 目录，比如 for java applications/party/src/org/ofbiz/party/party/PartyServices.java 和 for minilang applications/party/script/org/ofbiz/party/party/PartyInvitationServices.xml.各自的类路径和文件路径将在服务定义中体现。
*Step - 2 :在控制器中你必须创建一个入口为执行服务的请求和设置响应，象

```
<request-map uri="createPracticePerson">
    <security https="true" auth="true"/>
    <event type="service" invoke="createPracticePerson"/>
    <response name="success" type="view" value="PersonForm"/>
</request-map>
```

Step - 3: 现在所有写好的服务必须在服务器启动时装载，因此在 ofbiz-component.xml 文件中有一个服务定义的条目，如下：

```
<service-resource type="model" loader="main" location="servicedef/services.xml"/>
```

因此你无论在任何服务定义中的修改必须重启服务器才能生效。

为会员（Party）实体写 CRUD 操作：

开始我们将为会员写服务，然后当写创建 Person 服务时将调用会员的服务。
Step - 1 :在 servicedef 目录中创建一个文件"services.xml"
Step - 2 : 为 Party 实体定义 CRUD 操作的服务。服务的名称是 createPracticeParty, updatePracticeParty, deletePracticeParty，指定正确的位置到这些服务实现的文件，比如 /framework/example/script/org/ofbiz/example/example/ExampleServices.xml.
Step - 3 : 为这些服务的实现在你的组件目录中创建目录结构和 PracticeServices.xml 文件。
(实现可参考 Example 组件的 services.xml 和 ExampleServices.xml 文件。)
注:不要使用<override>标签在教程随后的介绍中。
从这里开始，你想运行这些服务的话，可以通过 webtools 运行这些服务：webtools--> Run Service . 这样你可以测试你的服务。
注: 你应该阅读 http://www.nabble.com/The-fancy-new-entity-auto-service-execution-engine-td18674040.html. 这功能近来已经增加，与对实体写 CRUD 操作的传统方式不同。
新功能让你仅定义这些服务，通过说起你想运行的操作。基本上设计引擎属性为"entity-auto"和调用(invoke)属性为"create", "update", or "delete"。你可以看下 example 组件的 services.xml 中的下列代码：

```
<service name="createExample" default-entity-name="Example" engine="entity-auto"
invoke="create" auth="true">
    <description>Create a Example</description>
    <permission-service service-name="exampleGenericPermission"
main-action="CREATE"/>
    <auto-attributes include="pk" mode="OUT" optional="false"/>
```

```
    <auto-attributes include="nonpk" mode="IN" optional="true"/>
    <override name="exampleTypeId" optional="false"/>
    <override name="statusId" optional="false"/>
    <override name="exampleName" optional="false"/>
</service>
```

engine="entity-auto" invoke="create"为缺省实体"Example."做创建记录的角色。
作为练习你可以做更进一步，这些步骤在帮助你理解概念，你仅能在你的代码中练习上面所给模式。

为个人 Person 实体写 CRUD 操作:

-为个人 person 实体创建记录需要有 partyId，因此首先调用 createPracticeParty 服务，然后再得到 partyId 后创建 person 记录。
- 我们将在 person 实体列表表单下面增加一个附加的表单。这个表单将调用这个服务来创建一个 person 记录。
Step - 1 : 创建附加表单来创建 person，把下面的 person 表单加在同一个 screen 中。

```
<form name="CreatePerson" type="single" target="createPracticePerson">
      <auto-fields-service service-name="createPracticePerson"/>
      <field name="submitButton" title="Create" widget-style="smallSubmit"><submit button-type="button"/></field>
</form>
```

Step - 2 : 为 person 实体写 CRUD 操作，这个就是在 services.xml 中 createPracticePerson 的代码。

```
<service name="createPracticePerson" default-entity-name="Person" engine="simple"
          location="component://practice/script/org/hotwax/practice/PracticeServices.xml" invoke="createPracticePerson" auth="true">
     <description>Create a Person</description>
     <auto-attributes include="pk" mode="OUT" optional="false"/>
     <attribute name="salutation" mode="IN" type="String" optional="true"/>
     <attribute name="firstName" mode="IN" type="String" optional="false"/>
     <attribute name="middleName" mode="IN" type="String" optional="true"/>
     <attribute name="lastName" mode="IN" type="String" optional="false"/>
     <attribute name="suffix" mode="IN" type="String" optional="true"/>
</service>
```

相似的 Update 和 Delete
Step - 3: 用可编辑的字段转换 List 表单 (参考： ListExampleItems from ExampleForms.xml) ，在里面加入更新

和删除选项，也把附加的表单加入同一 screen 中。
Step - 4 : 为这些服务增加控制项，是通过这表单来调用的。

写事件 Writing Events:

事件可以由 Java 和 minilang 来编写。下面的开发将写这些事件。
事件使用 Simple Map Processor 用来验证和转换。The Simple Map Processor Mini-Language 执行两个主要任务：验证和转换。在一个 context 从一个 Map 移到另一个。输入的 map 通常包括 Strings,但是能包含其它对象类型比如 Integer, Long, Float, Double, java.sql.Date, Time, and Timestamp。
在转向其它之前一个要浏览的重要链接是:
http://docs.ofbiz.org/display/OFBIZ/Mini-Language+Guide#Mini-LanguageGuide-smap。完成理解和实现可按下列步骤：
Step - 1 : 为此在你的实践应用菜单条中创建另一个标签(tab)，命名为"Events".
Step - 2 : 在 PracticeMenus.xml 文件中创建另一个带有两项的"EventMenu"菜单。这个菜单有 2 个菜单项，分别是"EventMinilang"和"EventJava"。一个在表单中用来调 minilang 的事件，另一个用来调 java 事件。
Step - 3 : 简单显示表示在两个请求上，用来创建一个新 person. 两个表单在调用目标事件是不同的。

```
<menu name="EventMenu" default-menu-item-name="eventMinilang"
default-selected-style="selected"

        type="simple" menu-container-style="button-bar button-style-1"
selected-menuitem-context-field-name="headerItem">

    <menu-item name="eventMinilang" title="Create Person --- Event MiniLang">

        <link target="createPracticePersonEventM"/>

    </menu-item>

    <menu-item name="eventJava" title="Create Person --- Event Java">

        <link target="createPracticePersonEventJ"/>

    </menu-item>

</menu>
```

Step - 4 : 在表单的屏幕上显示标签如"New Person - Simple Event"和"New Person - Java Event"，这样容易指出表单的不同目的。
Step - 5 : 在表单的目标中设置事件和在控制器中为事件创建请求映射。
一个要注意的重要事件 simple 事件控制项将象如下所示：

```
<request-map uri="createPracticePersonSimpleEvent">

    <security https="true" auth="true"/>{color}

    <event type="simple" path="org/hotwax/practice/PracticeEvents.xml"
invoke="createPracticePersonSimpleEvent"/>

    <response name="success" type="view" value="CreatePracPersonSimpleEvent"/>

    <response name="error" type="view" value="CreatePracPersonSimpleEvent"/>
```

```
</request-map>
```

这个路径是所写事件的文件路径，是 practice/script/org/hotwax/practice.
，java 事件控制器入口如下：

```
<request-map uri="createPracticePersonJavaEvent">
    <security https="true" auth="true"/>
    <event type="java" path="org.hotwax.practice.PracticeEvents"
invoke="createPracticePersonJavaEvent"/>
    <response name="success" type="view" value="CreatePracPersonJavaEvent"/>
    <response name="error" type="view" value="CreatePracPersonJavaEvent"/>
</request-map>
```

这个路径是事件定义的类路径（classpath）。
文件名是 PracticeEvents.java，创建在 practice/src/org/hotwax/practice.

简单事件 Simple Event

Step - 6 : 在 script/org/hotwax/practice/目录中创建一个文件 PracticeEvents.xml。
Step - 7 : 在 PracticeEvents.xml 文件中写事件 createPracticePersonSimpleEvent.(作为参考你可以从 party 组件的 UserEvents.xml 中浏览"createUser"事件)
你要写的事件应该是较简单的一个，仅需要处理 5 个字段：salutation, firstName, middleName, lastName, suffix. ，然后调用 createPracticePerson 服务。
处理这字段你将使用先前看到的 simple map processor。
按下列步骤写事件：
7.a : 象如下从表单中处理字段：

```
<call-map-processor in-map-name="parameters" out-map-name="createPersonContext">
    <simple-map-processor name="createPersonMap">
        <process field="firstName">
            <copy/>
            <not-empty>
                <fail-property property="PracticeFirstNameMissingError"
resource="PracticeUiLabels"/>
            </not-empty>   
        </process>
    </simple-map-processor>
</call-map-processor>
<check-errors/>
```

7.b： 在 fail-property 中创建象 PracticeFirstNameMissingError 用来显示的界面标签。

7.c： 调用服务 createPracticePerson，传递从处理字段中获取的 map 作为输入 map 给服务。

输出屏幕（OutPut Screen）：

Part - 4

Java 事件:

将写的 java 事件是非常简单的。你可以参考任意*Events.java 文件。.

Step - 1： 内容将是：

```
public static String createPracticePersonJavaEvent(HttpServletRequest request,
HttpServletResponse response){
    LocalDispatcher dispatcher = (LocalDispatcher)
request.getAttribute("dispatcher");
    GenericDelegator delegator = (GenericDelegator)
request.getAttribute("delegator");
}
```

Step - 2 : 你必须处理来自表单的字段

```
String salutation = (String) request.getParameter("salutation");
String firstName = (String) request.getParameter("firstName");
```

Step - 3 :为这些值准备一个 map，这个是传递给"createPracticePerson"服务。象

```
Map createPersonCtx = UtilMisc.toMap("salutation", salutation, "firstName",
firstName);
```

Step - 4 : 然后在后面仅仅象下面调用服务"createPracticePerson"

```
try{
    Map person = dispatcher.runSync("createPracticePerson", createPersonCtx);
 }catch (GenericServiceException e){
     Debug.logError(e.toString(), module);
     return "error";
 }
return "success";
```

在用 java 写完事件后，别忘了用 ant 命令编译。这时你必须把 build.xml 文件加到你的组件目录中，比如在 hot-deploy/practice/。关于 build.xml 文件内容你可以参考"example"组件。
在 build.xml 文件确保每一项都有下列条目：

```
<target name="classpath">
    <path id="local.class.path">
        <fileset dir="../../framework/base/lib/j2eespecs" includes="*.jar"/>
    </path>
</target>
```

需要如下的一些类：

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

创建一个 build.xml 文件然后进行编译。在编译后在你的组件中会创建一个目录，包含所有的*.jar and class 文件。build.xml 文件的内容你可以参考 example 组件。
要运行 simple 事件，不要忘记在 ofbiz-component.xml 增加条目<classpath type="dir" location="script"/>。要运行 java 事件，不要忘记在 ofbiz-component.xml 增加条目<classpath type="jar" location="build/lib/*"/>。

ECA(Event Condition Action):

ECA：它联合三件事：一个事件、每个事件的条件和每个事件的动作。 是一个规则用来在一个事件上当某条件满足时触发一个动作。当一个服务被调用比如执行一个查询会查看是否有这个事件的 ECA。事件包括执行前权限验正、输入参数校验前、在实际服务调用前、在输出参数校验前、在事件提交前或在服务返回前。接着在 ECA 定义中的每个条件被检查，如果返回真，则每个条件都会执行。一个动作也仅仅是一个服务，它定义的参数必须已经在服务的 context 中。条件或动作的数据在每个 ECA 定义中没有限制。
想了解详细请访问：Service Engine Guide(http://docs.ofbiz.org/display/OFBTECH/Service+Engine+Guide)
1. **SECA (Service Event Condition Action)** :这是用在执行一个服务时当某条件满足时触发另一个服务（动作）
2. **EECA (Entity Event Condition Action)**：这是用在创建一个实体记录时当某条件满足时触发另一个服务（动作）
再实现 ECA 也将按跟 screens 和 menus 相同的方法来按下列步骤：
Step - 1： 增加更多的应用菜单"ECA"到 practice 应用菜单条中.(在 controller.xml 加入必要的条目作为目标)
Step - 2： 在 PracticeMenus.xml 文件中创建另一个名叫"EcaMenu"的菜单。这个菜单有两个菜项："seca"和"eeca". 为每个菜单，两个 screens 都需要使用先前创建的"CreatePerson" (在 personForm screen)

```
<menu name="EcaMenu" default-menu-item-name="seca" default-selected-style="selected"
       type="simple" menu-container-style="button-bar button-style-2"
selected-menuitem-context-field-name="headerItem">
      <menu-item name="seca" title="Create Person --- SECA">
          <link target="seca"/>
      </menu-item>
      <menu-item name="eeca" title="Create Person --- EECA">
          <link target="eeca"/>
      </menu-item>
</menu>
```

SECA :

Step - 1： 你必须写另一个服务"createPartyRoleVisitor",它用来在运行"createPracticePerson"服务创建会员时设置会员角色。这个服务在"createPracticePerson"服务定义的 seca 规则时触发。
在新的服务中涉及实体"PartyRole". 在"createPartyRoleVisitor"中仅调用已经实现的服务"createPartyRole"。
Step - 2： 现在必须在"servicedef"目录中创建一个文件"secas.xml". Seca 定义如下(参考"party"组件的 secas.xml). 将会是

```
<eca service="createPracticePerson" event="commit">
    <condition field-name="partyId" operator="is-not-empty"/>
    <action service="createPartyRoleVisitor" mode="sync"/>
```

```
</eca>
```

Step - 3 :在 ofbiz-component.xml 文件中增加一条目作为 seca 定义的加载。将是:
<service-resource type="eca" loader="main" location="servicedef/secas.xml"/>
不要忘记在做这些后重启服务器。现在通过表单运行服务然后检查 PartyRole 实体的记录。你将发现创建的会员已设置一个角色，因为你通过 seca 规则同步触发另一服务为创建的会员设置角色。

EECA :

Step - 1 : 你必须写另一个服务"createPartyRoleCustomer",用来为创建的会员设置角色，意思是创建会员实体记录时这个服务被触发，从而为这个会员设置一个 role customer. 这个服务"createPartyRoleCustomer"与"createPartyRoleVisitor"相似。
Step - 2 : 现在必须在"servicedef"目录中创建一个文件"eecas.xml",这个目录是在你的组件目录"practice"中.
Eeca 定义也是在这(参考"accounting"组件的 eecas.xml)。将会是：

```
<!-- To create party role whenever a party is created -->
<eca entity="Party" operation="create" event="return">
    <condition field-name="partyId" operator="is-not-empty"/>
    <action service="createPartyRoleCustomer" mode="sync"/>
</eca>
```

Step - 3 : 在 ofbiz-component.xml 文件中增加一条目作为 eeca 定义的加载。将会是:

```
<entity-resource type="eca" reader-name="main" loader="main"
location="entitydef/eecas.xml"/>
```

不要忘记在做这些后重启服务器。现在通过表单运行服务然后检查 PartyRole 实体的记录。你将发现创建的会员已设置一个角色，因为你通过 eeca 规则同步触发另一服务为创建的会员设置角色。
主要不同点是当在实体上进行操作时触发一个服务。在我们的案例中是创建("create")。
注: 你已经创建各自的菜单来分别理解这概念。一旦你为服务"createPracticePerson"编写 seca,因而无论在你的 practice 应用哪儿调这服务时，seca 都将触发"createPartyRoleVisitor"，换句话说当一会员被创建时，"createPartyRoleCustomer"将被触发。
输出屏幕（Output Screen） :

组服务 Group Service:

组服务常用来调超过一个服务作为一组。服务组是可以一块运行的服务集合，可以通过调初始服务。你定义一个服务使用组服务引擎（group service engine），然后在这个组中包括所有服务的参数和属性。对于组服务来说位置属性是不必要的，invoke 属性定义运行这个组的名字。.当这个服务被启动，这个组被调用；在组中定义的服务按照规定被调用。

更多请看: http://docs.ofbiz.org/display/OFBTECH/Service+Engine+Guide

实现组服务按照下列步骤：

Step - 1 :加入一个菜单项到应用菜单条中，命名为"Group Service".(在 controller.xml 文件中要有必要的条目)

Step - 2 : 现在创建新的 screen 和一个表单来创建 person，因为表单的目标是定义的组服务。

注 : 现在该定义组服务了。我们将定义 practice 应用中已实现服务的组服务。

Step - 3 : 你将在 services.xml 文件中定义组服务。(参考会员(party)组件的 services.xml)。仅写一个或更多服务用来由 createPracticePerson 服务创建的会员设置"CLIENT"角色。

象下面那样创建一个"partyGroup"组服务：

```
<!-- Group service -->
<service name="partyGroup" engine="group" auth="true">
    <description>Creates a party, person and party role Client</description>
    <group>
        <invoke name="createPracticePerson" result-to-context="true"/>
        <invoke name="createPartyRoleClient"/>
    </group>
</service>
```

再测试前别忘了要重启服务器（server）。

接口 Interface:

实现接口服务引擎(interface service engine)来帮助定义的服务共享相同的参数。一个接口服务不能被调用，但是用来被其它服务来继承的。每一个接口服务使用接口引擎来定义。

更多信息请看： http://docs.ofbiz.org/display/OFBTECH/Service+Engine+Guide

实现这个按下列步骤：

Step - 1 : 加入一个菜单项到应用菜单条中，命名为"Interface".(在 controller.xml 文件中要有必要的条目)

Step - 2 : 现在创建新的 screen、一个表单和服务来创建。这个服务将实现这个接口。 (创建接口参考 accounting 组件的 services_fixedasset.xml) 如下所示：

```
<!-- Peson Interface -->
<service name="createPersonInterface" engine="interface" location="" invoke="">
    <attribute name="firstName" mode="IN" type="String" optional="false"/>
    <attribute name="middleName" mode="IN" type="String" optional="true"/>
    <attribute name="lastName" mode="IN" type="String" optional="false"/>
    <attribute name="suffix" mode="IN" type="String" optional="true"/>
</service>
<service name="createPracticePersonInterfaceService" engine="simple"
```

```
        location="org/hotwax/practice/PracticeServices.xml"
invoke="createPracticePersonInterfaceService" auth="false">
    <description>Creates a new Person</description>
    <implements service="createPersonInterface"/>
    <attribute name="partyId" mode="OUT" type="String" optional="false"/>
    <override name="suffix" optional="false"/>
</service>
```

这儿我们实现一个接口并覆盖后缀(suffix)属性的行为,在这个服务运行中生效。服务 createPracticePersonInterfaceService 的实现和 createPracticePerson.相同。不要忘记在这个实现后重启服务器。

Part - 5

创建新实体:

新实体的创建你可以再参考 example 组件，为此你可以看下 example 组件的 entitymodel.xml 文件。你可以按下列步骤创建新的实体：

Step - 1 : 在 hot-deploy/practice/.创建一个 entitydef 的子目录。

Step - 2 : 创建一个 entitymodel.xml 新文件。这个文件包括你想定义的实体定义。

Step - 3 : 要装载定义，你需要在 ofbiz-component.xml 文件中定义一个条目如下：

```
<entity-resource type="model" reader-name="main" loader="main"
location="entitydef/entitymodel.xml"/>
```

这里隐含你必须重启服务器才能让那些修改生效。

这里重点阅读：http://docs.ofbiz.org/display/OFBTECH/General+Entity+Overview.

你很少要找方式去定义新的实体，因为你已经在 OFBiz 有这些实体定义，对你的业务处理有很大的指导作用。虽然你可能感觉在某些地方要给存在的实体增加更多字段，你能如何做那些呢？下一步将告诉你方法，为你的客户化需要扩展一个实体。

先前我们通常在相同目录中需要一或更多 entitygroup.xml 文件，这个不再需要，因为代码已经检入到 trunk 中（because code is checked in to the trunk for this）.

扩展存在的实体:

是的，你可以扩展已存在的实体满足你客户化的需要。按下列方式完成：

Step - 1 : 在你的客户化应用的 entitydef/entitymodel.xml 文件中按这方式扩展你的实体。

```
<extend-entity entity-name="">
    <field name="" type=""/>
</extend-entity>
```

作为例子，你可以参考 party 组件的 entitymodel.xml 文件。
这是一个最简单的表单也能变得更复杂。这将增加一个或更多字段到已存在的实体中。它依赖你想要进行客户需求的字段。这儿你也能定义与其它实体的关系。但在做这些之前，你得特别地搜索下，可能你要加的字段作为其它目的已经存在，因此是简明的。也要去进行数据模型的扩展学习然后再做。
实体引擎配置别忘了看： Entity Engine Configuration Guide(http://ofbiz.apache.org/docs/entityconfig.html#Entity_Model)

为客户化的应用准备数据:

你的 practice 应用准备数据可按下列步骤：
Step - 1 : 在 practice 创建一个目录"data"，在这创建一个 PracticeData.xml 文件。
Step - 2 : 我们给一个用户创建数据，我必须为一个会员按指定顺序准备好，如下：

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

<entity-engine-xml>

    <Party partyId="DemoUser" partyTypeId="PERSON"/>

    <Person partyId="DemoUser" firstName="Practice" lastName="Person"/>

    <PartyRole partyId="DemoUser" roleTypeId="VISITOR"/>

    <ContactMech contactMechId="5000" contactMechTypeId="EMAIL_ADDRESS"
infoString="practice.person@gmail.com"/>

    <PartyContactMech partyId="DemoUser" contactMechId="5000" fromDate="2001-05-13
00:00:00.000" allowSolicitation="Y"/>

    <PartyContactMechPurpose partyId="DemoUser" contactMechId="5000"
contactMechPurposeTypeId="PRIMARY_EMAIL" fromDate="2001-05-13 00:00:00.000"/>

</entity-engine-xml>
```

目的是创建一个带有 VISITOR 角色的个人会员，然后为那个会员创建一个电子邮件地址作为主要的电子邮件地址。
Step - 3: 现在我们必须在 ofbiz-component.xml 中加入一个如下条目：

```
<entity-resource type="data" reader-name="demo" loader="main"
location="data/PracticeData.xml"/>
```

在完成这些后，你运行命令 ant run-install 会导入 demo 数据，从这个文件的数据将作为 demo 数据导入。一旦你启动服务器，在 practice 应用的 Person 表单里你能看到为个人增加的记录，或者你可以访问 https://localhost:8443/webtools/control/entitymaint ，查找每一个实体检查这些记录是否加入到数据库中。

结论:

如果你按照所有步骤从这个应用中开发 practice 应用，这将真正会帮助你我理解在 OFBiz 中的其它实现。这些是在 OFBiz 上工作的基础。现在你知道如何在 OFBiz 开始开发。不要漏掉在这个教程中的外部链接，他们将真正更多地帮助你理解这儿发生的事情。另一个能帮助你的好资料在 FAQ Tips Tricks Cookbook HowTo(http://docs.ofbiz.org/display/OFBIZ/FAQ+-+Tips+-+Tricks+-+Cookbook+-+HowTo)
接下来的事情就是真实的业务处理，如此这些书将非常有用：OFBiz Related Books (http://docs.ofbiz.org/display/OFBADMIN/OFBiz+Related+Books/)